databases

grondslagen voor de logische structuur

drs. F. REMMEN

ACADEMIC SERVICE

DATABASES

GRONDSLAGEN VOOR DE LOGISCHE STRUCTUUR

databases

grondslagen voor de logische structuur

drs. F. REMMEN

Academic Service
den haag 1982

Uitgegeven door: Academic Service
Postbus 96996
2509 JJ Den Haag
Zetwerk: mevr. I. Geerling-Engelbarts
Ontwerp omslag: JAM Gauw
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam
ISBN: 90 6233 081 9

VOORWOORD

Dit boek is gericht op de behandeling van de grondslagen voor de logische structuur van databases. Het bestaat uit drie delen. Het eerste deel is bedoeld als een eerste kennismaking met databases en hun omgeving. Daarin wordt nagegaan hoe het fenomeen database en het daaromheen hangende jargon is ontstaan en welke problemen om een (formele) oplossing vragen.

Het tweede deel is het meest omvangrijke. Het heeft de naam verzamelingsmodel meegekregen. Velen zijn het erover eens, dat de verzamelingsleer van groot nut kan zijn bij de oplossing van databaseproblemen. Om dit echter gestalte te geven in een adequaat formeel model, is blijkbaar toch niet zo eenvoudig. De afgelopen jaren heb ik namelijk bij cursussen in het hoger onderwijs en daarbuiten moeten ervaren, dat het een illusie is te veronderstellen dat 'modern' opgeleide personen een behoorlijke kennis van verzamelingsleer hebben, laat staan zulke kennis kunnen gebruiken. Men moet dan ook niet verwachten, dat toepassing van verzamelingsleer bij het oplossen van databaseproblemen een eenvoudige zaak is. Het is echter wel mogelijk, maar dan dient het te geschieden via een streng formele en systematische opzet. De ervaringen van de afgelopen jaren hebben geleerd, dat menig student in het begin flink moet worstelen met de materie van het tweede deel, maar achteraf dan wel kan vaststellen, dat hij daarmee een solide en werkzaam begrippenkader ter beschikking heeft gekregen. Vrijwel alle definities in dit tweede deel worden gegeven in twee vormen: niet formeel en formeel. De niet-formele vorm is bedoeld als een 'steuntje in de rug' voor het begrijpen van de formele vorm. Dat dit steuntje geen overbodige luxe is (gebleken) hangt samen met het boven gesignaleerde gebrek aan kennis van verzamelingsleer.

In het derde deel wordt nagegaan hoe de logische structuur van het verzamelingsmodel kan worden weergegeven in een belangrijke klasse van database-managementsystemen, namelijk die welke gebaseerd zijn op het zogeheten netwerkmodel. Evaluatie van dit model vindt plaats met behulp van de theorie uit het tweede deel.

Gezien de doelstelling van dit boek, grondslagen voor de logische structuur, mag beslist niet verwacht worden dat in dit boek 'alles te

vinden is' over databases. De lezer zij gewaarschuwd. Er is veel dat niet of maar in zeer beperkte mate in dit boek staat: vrijwel niets over de fysieke organisatie van de gegevens, niets over het hiërarchisch model, niets over methoden, heel weinig over beschikbare vraagtalen, niets over privacy-problemen. Ik zou zo nog wel even kunnen doorgaan. Men versta mij wel. Genoemde onderwerpen zijn beslist niet onbelangrijk. De reden echter, dat zij in dit boek niet of nauwelijks worden behandeld is, dat een enigszins volledige uitwerking van bovengenoemde doelstelling een goed afgerond geheel vormt, waar men blijkens de ervaringen van de afgelopen jaren 'reeds de handen vol aan heeft'. Ik hoop dan ook, dat de lezer in dit boek ook zaken zal aantreffen, die elders niet of zeker niet in die mate te vinden zijn. Dit geldt dan met name voor het tweede deel en voor de wijze waarop behandeling en evaluatie van het netwerkmodel plaatsvindt.

De meeste hoofdstukken zijn voorzien van een groot aantal opgaven. Voor het verwerven van een goed begrip van de geboden stof kunnen deze opgaven een uiterst nuttige functie vervullen.

In hoofdstuk 7 wordt een (naar gangbare onderwijsbegrippen) omvangrijke database gedefinieerd. Deze biedt door het relatief grote aantal objecten en hun onderlinge verbanden veel mogelijkheden om de echte databaseproblemen, namelijk die van massa en complexiteit, aan bod te laten komen.

Voor de samenstelling van dit boek ben ik veel dank verschuldigd aan collega's en studenten. Hun talloze vragen en opmerkingen hebben tot vele aanvullingen en verbeteringen geleid.

Een apart woord van dank is op zijn plaats aan Frans Groen van de BIO-Arnhem. En tenslotte wil ik een heel bijzonder woord van dank richten aan Bert de Brock van de TH-Eindhoven, voor met name de uitgebreide en deskundige medewerking aan de totstandkoming van het tweede deel.

Voor op- en aanmerkingen houd ik mij uiteraard gaarne aanbevolen.

Eindhoven, juli 1982. F. Remmen

INHOUD

DEEL I
KENNISMAKING MET DATABASES EN HUN OMGEVING

1 DATABASES EN HUN OMGEVING

1.1 HISTORISCHE ONTWIKKELING VAN GEGEVENSSTRUCTUREN

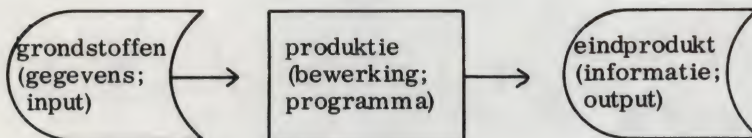
Om een organisatie (overheidsinstelling of particulier bedrijf) goed te kunnen besturen, moeten op bepaalde momenten de nodige beslissingen worden genomen. Voor het nemen van deze beslissingen is *informatie* nodig (BEM, hfdst.1).

Deze informatie zal in het algemeen niet kant en klaar beschikbaar zijn. Er zal dan ook een speciaal produktiesysteem moeten zijn, dat zorgt dat de gewenste informatie wel beschikbaar komt. Zo'n speciaal produktiesysteem noemen wij een *informatiesysteem*.

Zoals in elk produktiesysteem kunnen wij ook in een informatiesysteem spreken van

- het gewenste produkt, in casu de *gewenste informatie*; hiervoor wordt ook dikwijls de benaming *informatiebehoefte* gebruikt;
- de grondstoffen, die nodig zijn voor het maken van het gewenste produkt, in casu de *gegevens (data)*;
- het proces, waarmee de grondstoffen worden verwerkt tot het gewenste eindprodukt, in casu het *programma*.

Deze drie elementen vindt men ook terug in het onderstaande schema (LUNREM, hfdst.1).



Alleen als de gegevens goed *geordend* zijn, kan er sprake zijn van een betrouwbaar en efficiënt informatiesysteem. Aan zo'n ordening van gegevens zal een goede *gegevensstructuur* ten grondslag moeten liggen.

Deze gegevensstructuur moet uiteraard ontleend zijn aan de doelstelling van het informatiesysteem. Met andere woorden: uit de informatiebehoefte moet worden afgeleid, welke structuur voor de gegevens geschikt is en met welk verwerkingsproces dan uiteindelijk

aan de informatiebehoefte zal worden voldaan.

Het probleem van gegevensstructurering was in de vijftiger en begin zestiger jaren nog (betrekkelijk) eenvoudig. Gezien de toenmalige (beperkte) technologische mogelijkheden, die alleen sequentieel toegankelijke achtergrondgeheugens toelieten, kon men voor informatiesystemen alleen maar denken aan eenvoudige gegevensverzamelingen.

Elk van deze gegevensverzamelingen bestond uit gegevenseenheden, die op dezelfde soort van zaken betrekking hadden, of, meer technisch uitgedrukt: elke gegevensverzameling bestond uit één bestand van records. Voorbeelden: werknemersbestand, artikelenbestand, klantenbestand.

Met de komst van direct toegankelijke achtergrondgeheugens ging het beeld zich wijzigen. Zoals bij zoveel ontwikkelingen, gold ook hier: meer mogelijkheden, meer (informatie)behoeften. Of liever: deze (verdergaande) behoeften waren voorheen wel latent aanwezig, maar werden niet expliciet naar voren gebracht vanwege de (terechte) verwachting, dat er toch niet aan voldaan zou kunnen worden.

De informatiebehoeften, die met de voortgaande technologische ontwikkeling aan de oppervlakte kwamen, hadden met name betrekking op:

- verbanden tussen verschillende soorten gegevens, bijvoorbeeld tussen
  - -- klant- en ordergegevens in een handelsonderneming
  - -- afdeling- en patiëntgegevens in een ziekenhuis
  - -- docent-, student- en vakgegevens in een onderwijsorganisatie;
- het ('gelijktijdig') gebruik door verschillende gebruikers(groepen) van (gedeelten van) de totale gegevensverzameling. Zo zijn bijvoorbeeld
  - -- bedrijfsleiding en administratie geïnteresseerd in de artikelgegevens
  - -- specialisten en ziekenhuisadministratie geïnteresseerd in patiëntgegevens
  - -- docenten en studentenadministratie geïnteresseerd in studentengegevens.

Niet elke (groep van) gebruiker(s) hoeft uiteraard in dezelfde gegevens geïnteresseerd te zijn. Hierop komen wij nog uitgebreid terug (in de toelichting op het begrip horizontale onafhankelijkheid).

Met direct toegankelijke achtergrondgeheugens werd het mogelijk bovengenoemde verbanden tussen gegevenssoorten zodanig op te slaan, dat de gegevens ook redelijk toegankelijk werden volgens deze verbanden.

VOORBEELD. Met een niet-direct toegankelijk achtergrondgeheugen is het een vrijwel onbegonnen taak om bij een willekeurig klantenrecord uit het klantenbestand snel de bijbehorende orders uit het orderbestand ter beschikking te krijgen. Met een direct toegankelijk

achtergrondgeheugen is dit wel te verwezenlijken door bijvoorbeeld van het desbetreffende klantenrecord een ketting van bijbehorende orderrecords aan te leggen.

Naast het weergeven van genoemde verbanden in de gegevensstructuren, werden er ook pogingen ondernomen om in te spelen op de behoefte om alle gegevens van alle soorten tezamen weer te geven in één grote (gemeenschappelijke) gegevensverzameling. En dan duikt, eigenlijk ineens, de term *database* op. In juni 1963 is er in Santa Monica (VS) namelijk een symposium met de titel 'Development and management of a computer-centered Data Base' gehouden. Voor verdere historische informatie zij verwezen naar Olle (OLLE, hfdst.1).

In de loop van de zestiger jaren worden diverse (software management) systemen ontwikkeld, die een min of meer geïntegreerd gebruik van bestanden mogelijk maken, zonder dat de gebruiker/programmeur daarvoor uitgebreide eigen voorzieningen moet treffen. Voorbeelden zijn de systemen NIP S/FFS van IBM, IDS van General Electric, ADAM van de MITRE Corp., IDMS van System Development Corp., DISK FORTE van Burroughs. Voor meer en uitgebreide informatie hierover zij verwezen naar het desbetreffende rapport van het CODASYL Systems Committee (CODASYL 69).

In het algemeen kan men zeggen, dat elk van deze systemen ontstond als *uitbreiding* van bestaande (data management) systemen. Zulke uitbreidingen hebben echter meestal het nadeel, dat de overzichtelijkheid afneemt. Naar analogie van opgevoerde motoren zou men hier kunnen spreken van opgevoerde data management systemen. En zoals meer bij opgevoerde systemen blijkt, heeft men dan te maken met een instrument, dat eigenlijk niet helemaal geschikt is voor het verwezenlijkne van de (nieuwe) doelstelling. Men ging zich dan ook afvragen of de zaak niet grondiger moest worden aangepakt, namelijk eerst kijken naar het (gegevens)probleem en dan aangepast gereedschap ontwikkelen.

Met andere woorden: men diende meer de architectuur van een database als uitgangspunt te nemen. Dit gebeurde onder meer door de Data Base Task Group (afkorting: DBTG) van het CODASYL Programming Language Committee. Het is overigens interessant om op te merken en tekenend voor de historische context, dat de (oorspronkelijke) taak van de DBTG (was) is: "to define extensions to COBOL to handle databases".

Olle geeft een interessante schets over het ontstaan en de evolutie van de DBTG en andere groepen binnen CODASYL-verband (OLLE, § 1.3-1.5).

In 1971 bracht de DBTG een inmiddels welbekend rapport uit (CODASYL 71). Hierop zullen wij uitgebreid ingaan in het derde deel van dit boek. Dit rapport is bedoeld als een machine-onafhankelijk voorstel van taalelementen, nodig voor de beschrijving en het gebruik van een database.

Deze machine-onafhankelijke opstelling heeft echter niet verhin-

derd, dat in het DBTG-rapport taalelementen worden voorgesteld, die betrekking hebben zowel op het hoogste logische niveau van de gebruiker als op lager liggende niveaus van implementatie. Dit komt de doorzichtigheid uiteraard niet ten goede. Met name bij uitgebreide toepassingen wordt het daardoor moeilijk de diverse fasen in de opzet van een database duidelijk te onderscheiden.

Eind zestiger jaren/begin zeventiger jaren ontstond, vooral onder aanvoering van E.F. Codd, een streven om een model te ontwerpen, waarin de architectuur van een database op het logische niveau duidelijk naar voren komt (CODD 70). Hierbij wordt gebruik gemaakt van het wiskundige begrip relatie. Daarvandaan de naam *relationele* model. Eigenlijk is dit het begin geweest van een meer fundamentele, systematisch gerichte aanpak van ontwerp en gebruik van databases. Hetgeen overigens niet wil zeggen dat de (lawine van) literatuur over het relationele model de kwalificatie fundamenteel en systematisch verdient.

Met name de wiskundige ondergrond is op vele literatuurplaatsen erg zwak, met het gevolg dat het begrip van gegevensstructurering wazig wordt en het gebruik van gegevens ondoorzichtig niettegenstaande de zogeheten user-friendly query-languages.

Aan de fundamentele opbouw zal in dit boek veel aandacht worden geschonken. Het zal zelfs het belangrijkste en grootste deel van dit boek vormen. Omdat deze opbouw plaatsvindt met behulp van de verzamelingenleer, heb ik dit deel 'verzamelingsmodel' genoemd.

Om enig idee te hebben van de omgevingswereld van databases en het daarin gebruikte 'jargon', komen in dit inleidende hoofdstuk nog de volgende onderwerpen aan bod:
- omschrijving van het begrip database
- de organisatie rondom een database
- data independence en conceptual schema
- database modellen.

1.2 OMSCHRIJVING VAN HET BEGRIP DATABASE

In het volgende deel zullen wij tot een formele definitie van een zogeheten databasetype komen. In deze paragraaf zullen wij met behulp van een minder formele definitie trachten aan te geven, welke aspecten van belang zijn bij een database. We zullen in deze paragraaf verder kortheidshalve het woord definitie gebruiken, maar dan in de betekenis van niet-formele omschrijving.

Het aantal database-definities, dat in omloop is, is welhaast net zo groot als het aantal schrijvers over dit onderwerp, en dat is waarlijk niet gering. Met een variant op een bekend gezegde zou men in dit verband kunnen spreken van 'quot capita, tot definitiones'

('zoveel hoofden, zoveel definities'). Ik zal er hier enkele aanhalen (uit boeken, verschenen sinds 1975).

Knuth: "A large file or a group of files is frequently called a database."

Olle: "A database (is) a crossreferenced collection of data records of different types." (OLLE, 8). In tegenstelling tot de database wordt door Olle de file gedefinieerd als "a collection of records, which is not crossreferenced and in which the records are normally all of the same type" (OLLE, 8).

Palmer: "A widely accepted definition of a database is:
-"An organized, integrated collection of data;
- a natural representation of the data, with no imposed restrictions or modifications to suit a computer;
- capable of use by alle relevant applications, without duplication of data"".

Date: "A working definition of database (based on one given by Engles): A database is a collection of stored operational data used by the application systems of some particular enterprise." (DATE, 7).

Martin: "A database may be defined as a collection of interrelated data stored together without harmful or unnecessary redundancy to serve multiple applications." (MARTIN, 22).

Zonder deze definities in detail te bespreken (is ook weinig zinvol) kan volgens mij wel gesteld worden, dat Knuth erg onvolledig is, Palmer te wijdlopig en vaag en dat de definities van Olle, Date en Martin tezamen wel ongeveer weergeven wat de essentiële kenmerken van een database dienen te zijn.

Onderstaande definitie is bedoeld als een *poging* mijnerzijds om het begrip database duidelijk en volledig weer te geven.

> Een database is een verzameling van *permanente* gegevens, die ter beschikking staat van *alle* gebruikers van een informatiesysteem. Deze gegevens hebben betrekking op de voor het informatiesysteem *relevante objecten* en de bij deze objecten behorende *relevante kenmerken*.

Er volgt nu een nadere toelichting op (verschillende onderdelen van) deze definitie.

- Verzameling van *permanente* gegevens.
  Wij spreken van permanente gegevens, als deze ook *buiten* de programma's, die gebruik maken van deze gegevens en eventueel

zelfs de waarde daarvan wijzigen, *moeten* blijven bestaan (voor een bepaalde periode).
Niet-permanente (tijdelijke) gegevens zijn eigenlijk altijd mutatiegegevens. Dat tijdelijke gegevens dikwijls nog enige tijd bewaard blijven, is eigenlijk in principe overbodig, maar wel nodig, zelfs noodzakelijk, in verband met herstel van mogelijk optredende storingen en fouten.
Voorbeeld. Tot de permanente gegevens in een onderwijsorganisatie behoren onder meer naam, adres, woonplaats van elke student. Een adresmutatie is een tijdelijk gegeven.

- Deze permanente gegevens dienen ter beschikking te staan van *alle* gebruikers van een informatiesysteem.
  Deze bestemming voor vele gebruikers heeft belangrijke consequenties. Immers verschillende groepen gebruikers zullen op verschillende manieren geïnteresseerd zijn in de gegevens. Zo zal een administrateur in een ziekenhuisorganisatie behoefte hebben aan financiële gegevens over personen en tarieven, terwijl een specialist moet kunnen beschikken over (medische) gegevens van personen en van toe te passen behandelingen.
  Date (DATE, 7) spreekt van "the application systems of some particular enterprise". Per enterprise (een apart informatiesysteem en) dus een aparte database. Dit kan een onnodige beperking zijn. Informatiesystemen, en bijbehorende databases, die betrekking hebben op combinaties van 'gelijksoortige enterprises' bestaan reeds, zoals voor combinaties van ziekenhuizen. Systemen voor combinaties van 'ongelijksoortige' organisaties zijn in principe ook mogelijk, maar in de praktijk komen ze (bij mijn weten) (nog) niet voor. Misschien is dit in de toekomst wel te verwachten bij gebruik op grotere schaal van gedistribueerde gegevensverwerking.
  Er kan zelfs sprake zijn van (deels) tegenstrijdige belangen voor verschillende groepen. Met name kan dit optreden in verband met de gewenste gegevensopslag. Als bijvoorbeeld ordergegevens 'dichtbij' klantgegevens liggen opgeslagen ten gerieve van de factuurafdeling, dan is dat minder prettig voor de produktieafdeling, die de ordergegevens liever 'dichtbij' de fabricagegegevens heeft.
  Het ontwerp en de implementatie van een database zal daarom een compromis zijn tussen de verschillende (deels tegenstrijdige) verlangens van de verschillende gebruikers(groepen).
  Voor dit compromis is een 'neutrale instantie' nodig, in de vorm van een zogeheten DBA (Data Base Administrator). De DBA heeft dus tot taak
  - een redelijk compromis tot stand te brengen tussen de (tegenstrijdige) gebruikersbelangen. Hieruit volgt dat tot zijn verantwoordelijkheid o.a. hoort
    -- het definiëren en laden van de database

-- de toegangs- en onderhoudsstrategie van de database (wie mag wat gebruiken, wie mag wat veranderen)

De taak van een DBA is verre van eenvoudig. Data zegt dan ook terecht dat "the position of a DBA is a very senior one" (DATE, 10). Voor een uitgebreide taakbeschrijving van een DBA zij verwezen naar Date (DATE, 25-27).

In onderstaande tekening (van Mandy Kaas uit Eindhoven) is op ludieke wijze weergegeven hoe vele verschillende (groepen van) gebruikers verschillend tegen een database aankijken.



- De gegevens hebben betrekking op de voor het informatiesysteem *relevante objecten* en de bij deze objecten behorende *relevante kenmerken*.

  Hiermee raken wij aan een heel belangrijk onderwerp in de database wereld. Het heeft al heel wat verhitte discussies opgeroepen met meer scheiding dan vereniging der geesten als resultaat.

  Het ontwerpen van een adequaat formeel systeem voor het vastleggen van en manipuleren met relevante objecten en kenmerken, is een uiterst moeilijke en eigenlijk nog maar net op gang gekomen zaak. Het is dan ook te betreuren, als deze toch al niet eenvoudige zaak onnodig bemoeilijkt wordt door:

  a. het niet goed uit elkaar houden van de begrippen 'soort' en 'individu';
  b. de heilloze verwarring die ontstaat, door te stellen dat de

begrippen 'object' en 'kenmerk' uitwisselbaar zijn.

Elk van deze twee punten wordt nu nader besproken.

Ad a

Het verschil tussen soort en individu wordt in gangbare database terminologie meestal aangegeven met het verschil tussen *type* en *occurrence*. Men kan bijvoorbeeld spreken van het (object)type student, en van de (object)occurrences van alle studenten van een onderwijsinstelling. Het gaat in dit voorbeeld dan om één type (soort) met vele occurrences (individuen).

Codd heeft terecht opgemerkt, dat het niet goed onderscheiden van type en occurrence een bron kan zijn van vele onduidelijkheden en fouten.

Wij willen in dit verband nog wijzen op een belangrijke zaak, namelijk de wijze waarop (ten behoeve van onderlinge communicatie) een type en de occurrences van een type worden gerepresenteerd. Wij kunnen de benaming 'student' gebruiken voor het representeren van een objecttype.

Voor het representeren van één occurrence van dit type, dus van een individuele student, zullen wij waarden nodig hebben van één of meer kenmerken van deze individuele student, bijvoorbeeld JANSEN (waarde van kenmerk achternaam), ZEEPAD 15 (waarde van kenmerk woonplaats), EINDHOVEN (waarde van kenmerk woonplaats). Een andere representatie zou zijn met behulp van het kenmerk registratienummer, dat in dit geval bijvoorbeeld de waarde 101973 zou kunnen hebben.

Ad b

Volgens o.a. Smith & Smith kan een object eventueel ook als kenmerk optreden. Deze bewering wordt dan 'geschraagd' met het begrip 'objectrelativiteit' (SMITH, 42). Dit begrip klinkt aardig, maar maakt een duidelijke behandeling van gegevensstructuren erg moeilijk, zo niet onmogelijk.

Men is m.i. tot dit (wan)begrip gekomen, doordat men in de natuurlijke taal toegestane maar onnauwkeurige benamingen zonder meer heeft overgenomen in de formele beschrijving van gegevensstructuren. Een voorbeeld moge een en ander verduidelijken.

Stel gegeven is het object 'auto' met de kenmerken: kenteken, bouwjaar, kleur, eigenaar. Het is zeer wel denkbaar, dat in het informatiesysteem naast 'auto' ook 'eigenaar' een relevant object is, met bijvoorbeeld de kenmerken: naam, adres, woonplaats, telefoonnummer. Ogenschijnlijk treedt nu eigenaar op als kenmerk, maar ook als object. In feite wordt echter met het kenmerk 'eigenaar' bij het object 'auto' een kenmerk bedoeld, zoals 'naam van eigenaar'. Dit zal met name duidelijk worden als men een waarde van het kenmerk 'eigenaar' wil weergeven.

Naar aanleiding van het juist gegeven voorbeeld zij nogmaals gewezen op het reeds ad a genoemde onderscheid tussen type en

en occurrence. 'Auto is namelijk wel een naam van het onderhavige type, maar kan niet de naam zijn van een occurrence van dit type.

Na het bovenstaande moge het volgende uitgangspunt duidelijk zijn:

Een object kan nooit optreden als kenmerk en een kenmerk nooit als object.

Tenslotte moet nadrukkelijk worden opgemerkt, dat de relevantie van een object of een kenmerk voor een informatiesysteem *uitsluitend* kan worden bepaald door de gebruiker(s) en niet door degenen, die verantwoordelijk zijn voor de bouw van de database.

1.3 DE ORGANISATIE RONDOM EEN DATABASE

Om de objecttypen en kenmerktypen van een database te kunnen beschrijven, moet men kunnen beschikken over een zogeheten DDL (Data Description Language). (Vergelijk de taalelementen voor het declaratiegedeelte in ALGOL en PASCAL en voor de DATA DIVISION in COBOL.)

Een beschrijving van een database, zoals hierboven bedoeld, wordt ook wel een schema genoemd. Voor één bepaalde toepassing zal het als regel niet nodig zijn dat men de (beschrijving van de) gehele database, dus het totale schema, ter beschikking heeft. Daarom kan men dikwijls volstaan met deelbeschrijvingen, zogeheten subschema's.

Voor het onderhoud van de database (invoegen, veranderen, verwijderen van gegevens) en voor het gebruik van de gegevens in de database, moet men dan verder kunnen beschikken over een zogeheten DML (Data Manipulation Language). Een DML kan deel uitmaken, of beter gezegd een uitbreiding zijn van een reeds bestaande 'gewone' programmeertaal, zoals COBOL of FORTRAN. Men noemt die reeds bestaande taal dan een *host language*. Is een DML geen onderdeel van een host language, dan spreekt men van een *self contained* DML.

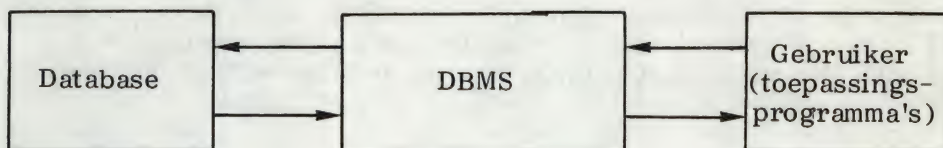
Voor een DDL kunnen analoge opmerkingen worden gemaakt.

Een waarschuwing voor mogelijke begripsverwarring is hier op zijn plaats. Zo wordt DML ook gebruikt als afkorting van Data Modelling Language, hetgeen hetzelfde is als de reeds eerder genoemde DDL. Voor Data Manipulation Language wordt ook gebruikt het (wat onduidelijke) begrip DSL (Data Sub Language) (DATE 6, 19). Om tenslotte de verwarring compleet te maken moet nog worden vermeld, dat DSL ook dikwijls wordt gebruikt als afkorting van Data Storage Language. Zo'n Data Storage Language is nodig om de opslagstructuur van de gegevens te beschrijven.

In het algemeen zal een DML vrij eenvoudig van opzet zijn (een gering aantal relatief eenvoudige taalelementen). Dit betekent niet dat onderhoud en gebruik van een database een eenvoudige zaak is.

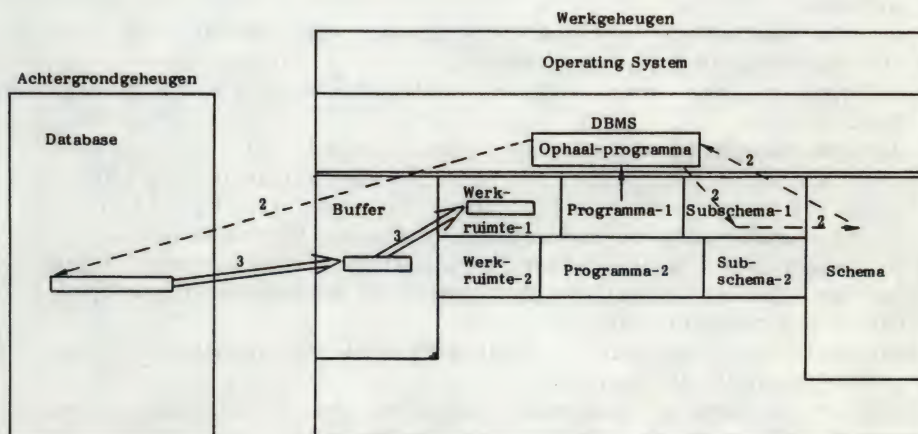
Het is echter niet de bedoeling de gebruikers via een ingewikkelde DML op te schepen met het gecompliceerde probleem van database benadering. Daarvoor is een uitgebreid pakket van speciale programmatuur beschikbaar, het zogeheten DBMS (Data Base Management System). "The DBMS is the software that handles all access to the database." (DATE, 25). Zo'n DBMS is een programmapakket met een niet ongebruikelijke orde van grootte van 50K woorden.

Elke gebruiker communiceert via het DBMS met de database (zie figuur 1.1).



Figuur 1.1 Plaats van een DBMS.

Een DBMS kan in feite worden beschouwd als een speciale uitbreiding van het operating system. Om de rol ervan nog iets duidelijker (ofschoon nog zeer schematisch) aan te geven, moge figuur 1.2 dienen, waarin in grote lijnen wordt aangegeven hoe een aanvraag vanuit een programma naar een bepaald gegeven uit de database via het DBMS tot stand komt.



Figuur 1.2 Verwerking van een ophaalopdracht m.b.v. een DBMS.

Hierbij is het volgende op te merken:

pijl 1 Ophaalopdracht in programma 1 wordt doorgegeven aan DBMS

pijlen 2 Met behulp van subschema en schema wordt de plaats van de gevraagde gegevens in de database bepaald

pijlen 3 De gevraagde gegevens worden via de bufferruimte getransporteerd naar de (eigen) werkruimte van het aanvragende programma.

Naast het relatief duidelijke begrip Data Base Management System komt men ook het begrip Data Base System (DBS) tegen. Volgens Date (DATE, 3) dient men daaronder te verstaan: de (fysieke) database zelf, dus de verzameling van de gegevens plus batch toepassingsprogramma's en terminal gebruikers. Wij zullen het begrip in een meer omvattende zin gebruiken, namelijk de (fysieke) database zelf en de hele organisatie (aan apparatuur en programmatuur en mensen), die nodig is voor onderhoud en gebruik van de database.

1.4 DATA INDEPENDENCE EN CONCEPTUAL SCHEMA

Bij het (structureren) vastleggen van gegevens kunnen wij spreken van verschillende elkaar opvolgende fasen. Hoeveel fasen men in feite wil onderscheiden, is op zich niet zo belangrijk. Immers, het onderscheid in fasen geschiedt om op deze wijze het vastleggen van gegevens op een overzichtelijke wijze tot stand te kunnen brengen. Ingewikkelde zaken kunnen nu eenmaal het beste stuksgewijs worden uitgevoerd.

Bij het vastleggen van gegevens zouden wij bijvoorbeeld de volgende fasen kunnen onderscheiden:

- de *logische* fase, waarin de zogeheten *logische structuur* wordt vastgelegd.
  Wat deze logische structuur precies inhoudt zal voor een groot gedeelte het onderwerp van de volgende hoofdstukken zijn. Intuïtief kunnen wij ons nu echter daarvan al wel enigszins een beeld vormen. Bij de logische fase denken wij namelijk aan het vastleggen van de (relevante) objecttypen en hun kenmerken, zonder daarbij ook maar enige aandacht te besteden aan opslag- en toegangsproblemen.
- de *opslag*fase, waarin de zogeheten *opslagstructuur* (Storage Structure) wordt vastgelegd.
  Men denke hier bijvoorbeeld aan zaken als: keuze van bestandsorganisatie voor de verschillende objecttypen, vaststellen van (maximaal te verwachten) occurrences per objecttype, wijze waarop directe toegankelijkheid (met bijvoorbeeld sleutelconversie)

gerealiseerd dient te worden.
In deze fase komt de fysieke weergave van de gegevens nog niet aan de orde; dit gebeurt in
- de *fysieke* fase, waarin de zogeheten *fysieke structuur* wordt vastgelegd. In deze fase wordt bijvoorbeeld bepaald welke opslagmedia zullen worden gebruikt.

Er werd reeds op gewezen dat deze indeling in fasen nog lang geen uitgekristalliseerde zaak is in de database-wereld. Hiervoor kunnen twee belangrijke oorzaken worden genoemd (die elkaar overigens beïnvloeden).

Ten eerste is de *theorie* over databases nog veel te weinig (wetenschappelijk) ontwikkeld om al genoeg ondersteuning te geven voor een duidelijke markering in fasen. Ten tweede is er bij de tot nu toe operationeel beschikbare database management systemen niet of veel te weinig sprake van splitsing in (opeenvolgende) fasen.

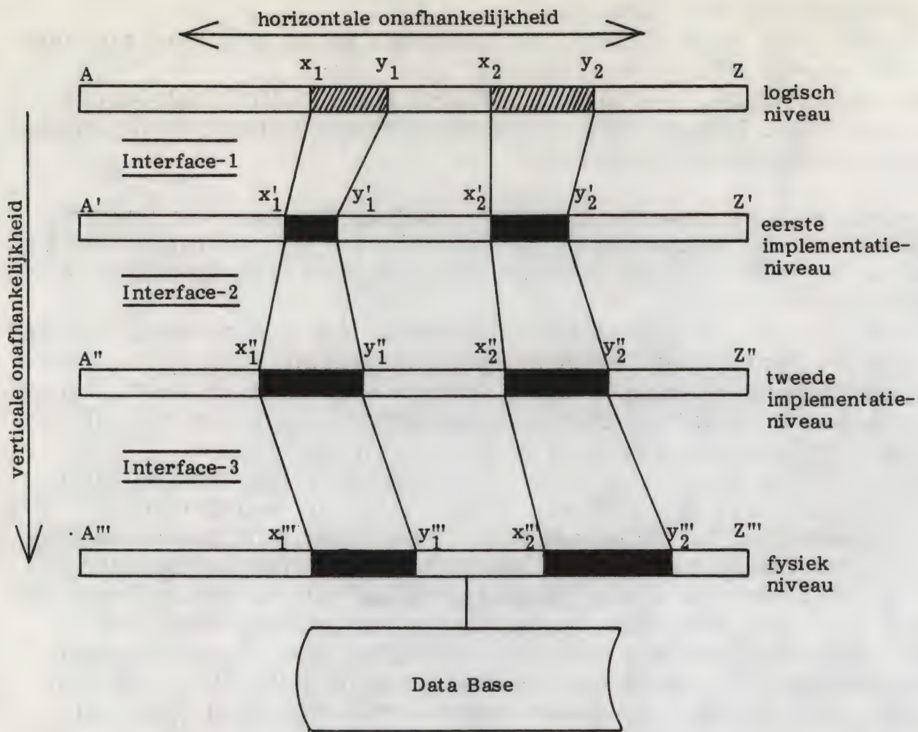
Zo komt het nogal eens voor, dat vanwege veranderingen in de database, die eigenlijk niet van belang zijn voor een gebruiker, deze toch genoodzaakt wordt (eventueel via de DBA) zijn gegevensdefinities en programma's te herzien met eventuele consequenties van opnieuw vertalen en laden. Als bijvoorbeeld een andere opslagvorm (dus een andere bestandsorganisatie) wordt gekozen voor een objecttype, dan moet een 'gewone' gebruiker daar eigenlijk 'geen last van hebben'. Deze afhankelijkheid van de gebruiker van voor hem eigenlijk irrelevante zaken, werkt irriterend en stagnerend.

Het is dan ook voor de hand liggend dat er wordt gestreefd naar zogeheten *data independence*. Over dit begrip willen wij ons nu een zo duidelijk mogelijk beeld vormen, inclusief de prijs die ervoor betaald moet worden.

Om een gebruiker te vrijwaren van bemoeienissen met 'vreemde' zaken is het goed zich te realiseren, dat de gebruiker
- alleen geïnteresseerd is in de logische (gegevens)structuur, dus niet in de implementatie van deze logische structuur;
- op het logische niveau verder alleen geïnteresseerd is in voor hem relevante zaken. Zo zullen in een ziekenhuisorganisatie administrateur en specialist verschillende interessegebieden hebben.

Vrijwaring van niet-relevante zaken op logisch niveau leidt tot zogeheten *horizontale data independence*. Voor de gebruiker is het dan als omvat het logische niveau niet meer dan voor hem relevante zaken (de onderdelen x_1-y_1 en x_2-y_2 in figuur 1.3).

Vrijwaring van diverse implementatiezaken leidt tot zogeheten *verticale data independence*. Voor de gebruiker is het dan alsof hij rechtstreeks kan opereren vanuit het hem bekende logische niveau, zonder dat daarvoor diverse implementatieniveaus nodig zijn. De niet-gearceerde vakjes van figuur 1.3 bestaan dus voor de gebruiker niet.



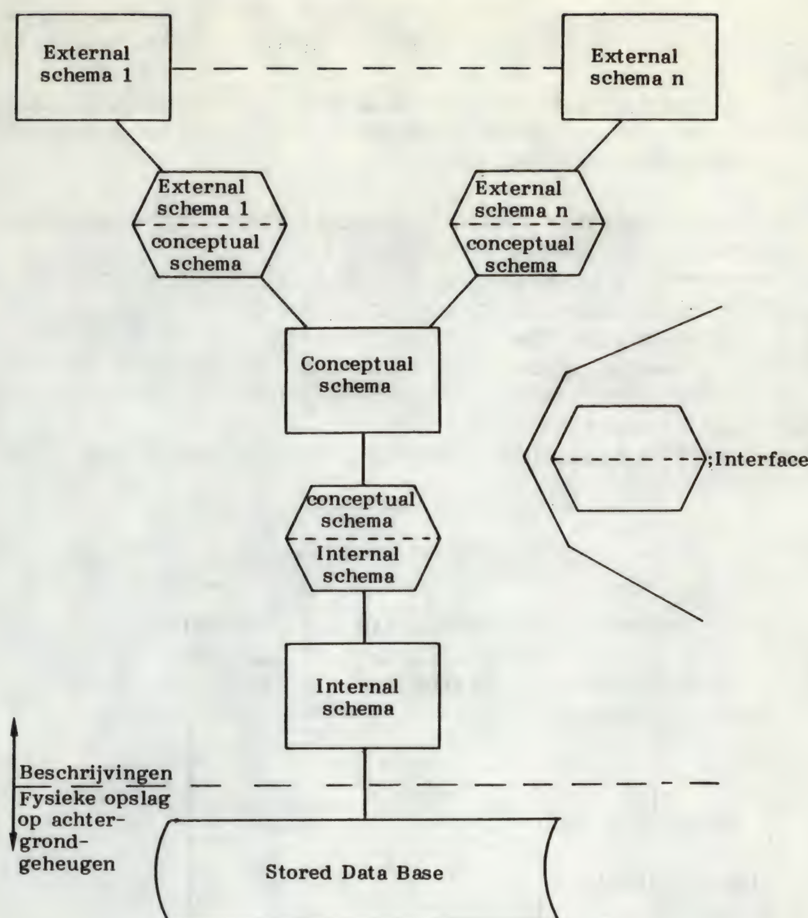
Figuur 1.3 Horizontale en verticale data independence.

In feite omvat de database meer dan de gebruiker zich in zijn door horizontale en verticale data independence afgeschermde omgeving realiseert. Om deze onafhankelijkheid tot stand te brengen, zullen dan ook maatregelen moeten worden getroffen.

Deze maatregelen hebben de vorm van een speciaal stuk software, zogeheten *interfaces*, die het verband leggen tussen de database wereld van de gebruiker en de totale database. *Interfaces zijn de prijs, die betaald moet worden voor data independence.*

De mate van onafhankelijkheid is evenredig met de interface-software, die ervoor ingezet wordt. Deze prijs moet men niet gering achten (stijgende software-prijzen tegenover dalende hardware-prijzen!).

In de literatuur (en in voorstellen voor standaardisatie) komt men nogal eens een afbeelding tegen, zoals in figuur 1.4. Deze afbeelding is dan bedoeld als een schets van de architectuur van een database systeem. Merk nogmaals op, dat een database systeem niet hetzelfde is als een database. Een database systeem omvat de database èn alles wat nodig is voor beschrijving en gebruik van de database.



Figuur 1.4 Architectuur van een database systeem.

In feite wordt in figuur 1.4 hetzelfde weergegeven als in figuur 1.3, als men bedenkt dat

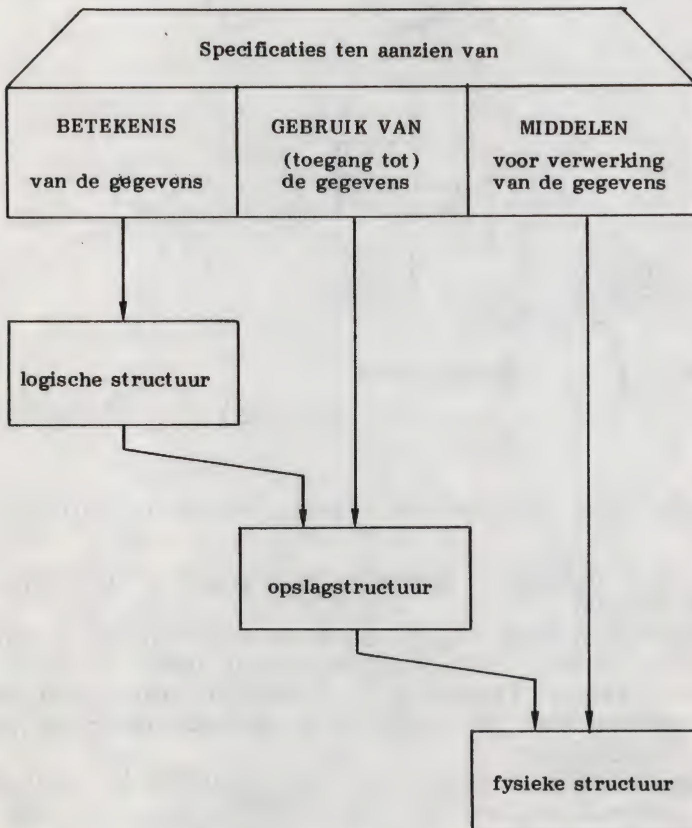
- het *conceptual schema* (figuur 1.4) de beschrijving is van de database op *logisch niveau* (A-Z in figuur 1.3);
- elk *external schema* (figuur 1.4) de beschrijving is van een *gegevensgebied* voor een bepaalde groep gebruikers (x_1-y_1 en x_2-y_2 in figuur 1.3);
- het *internal schema* (figuur 1.4) een afbeelding is van het conceptuele schema ten behoeve van efficiënte fysieke opslag. Dit is te vergelijken met het niveau A'''-Z''' in figuur 1.3).
  In figuur 1.4 zouden, evenals in figuur 1.3, ook meerdere implementatieniveaus kunnen worden weergegeven.

- de *stored database* (figuur 1.4) gelijk is aan de fysiek opgeslagen occurrences met bijbehorende 'administratie' (de 'database' in figuur 1.3);
- de onafhankelijkheid binnen en tussen de niveaus wordt waargemaakt met behulp van *interfaces* (in figuur 1.3 niet getekend voor horizontale onafhankelijkheid).

In dit boek zullen wij ons vooral bezighouden met het conceptuele schema, dus met de *logische structuur*.

In dit verband zij er nogmaals nadrukkelijk op gewezen, dat voor het bepalen van de logische structuur alleen de betekenis van de gegevens van belang is. De logische structuur zal op haar beurt van invloed zijn op de opslagstructuur, maar het is daarvoor *niet de enig* bepalende factor. Een analoge opmerking geldt voor de overgang van opslagstructuur naar fysieke structuur.

Een en ander is schematisch weergegeven in figuur 1.5.



Figuur 1.5 Fasering database-ontwerp.

Op deze figuur zullen wij in hoofdstuk 5 over normalisatie terugkomen, om duidelijk te maken dat normalisatie voor de logische structuur een heel geschikt begrip is, maar ongeschikt kan (en in het algemeen zal) zijn voor de opslagstructuur vanwege het feit dat 'betekenis van gegevens' en 'gebruik van gegevens' tot tegenstrijdige structuureisen kunnen leiden.

1.5 DATABASE MODELLEN

Als men een conceptueel schema wil opstellen en de bijbehorende manipulaties op de database wil vastleggen, doet zich uiteraard de vraag voor op welke wijze, met welk (taal)gereedschap men dit tot stand wil brengen. Dit gereedschap is eigenlijk zelf weer afhankelijk van de zienswijze die men heeft, met andere woorden afhankelijk van het database model.

De drie bekende modellen zijn:
- relationele model
- netwerk model
- hiërarchisch model.

Over deze modellen zijn in de loop der jaren (verhitte) discussies gevoerd. Voor een korte inleiding zij verwezen naar (DATE, hfdst.3) en (LUN-REM, hfdst.10).

In feite zijn deze modellen meer voorbeelden van bepaalde methoden van aanpak dan van database theorieën. Een uitzondering kan misschien worden gemaakt voor het relationele model, althans voor bepaalde onderdelen daarvan. Want ook daarin geldt, dat veel van het gebodene te maken heeft met een methode. Met name geldt dit als sprake is van een zogeheten *relational database*.

Met het bovenstaande wil niet gezegd zijn, dat bepaalde methoden niet erg nuttig zouden kunnen zijn. Maar om een methode te kunnen evalueren, dient men eerst wel een goede theoretische ondergrond te hebben. In dit boek is daartoe in het tweede deel een poging ondernomen onder de naam van verzamelingsmodel. In grote lijnen kan men dit verzamelingsmodel zien als een theorie, waartoe het relationele model de eerste stoot heeft gegeven, maar die dan verder consequent is opgebouwd met behulp van verzamelingenleer. In het derde deel van dit boek vindt dan een presentatie plaats van het netwerk model, met een evaluatie op basis van de voorgaande database theorie.

OPGAVEN

In het algemeen leent de stof van dit inleidende hoofdstuk zich niet voor opgaven behalve het onderwerp 'relevante objecten en kenmerken'.

1.1 Gegeven zijn van de organisaties onderwijsinstelling en handelmaatschappij onderstaande relevante objecten.

organisatie	relevante objecten
a. onderwijsinstelling	studenten docenten vakken afdelingen examens
b. handelmaatschappij	leveranciers klanten artikelen

Bedenk zelf relevante kenmerken bij elk van de objecten.

1.2 Bedenk zelf relevante objecten en kenmerken voor de volgende organisaties:
a. produktiebedrijf
b. gemeente
c. garagebedrijf.

DEEL II
HET VERZAMELINGSMODEL

2 WISKUNDIGE BASISBEGRIPPEN

2.1 VERZAMELINGEN

We beginnen met het ongedefinieerde begrip *verzameling* (Engels: set). Hierbij denken wij niet alleen aan verzamelingen zoals die van de natuurlijke getallen N, gehele getallen Z, rationele getallen Q en reële getallen R, maar vooral ook aan verzamelingen zoals: alle namen van ingeschreven studenten, alle combinaties van (registratienummer student, behaald cijfer) betreffende de uitslag van een bepaald tentamen.

Een verzameling is geheel bepaald door zijn elementen. Hieruit volgt dat elk element precies eenmaal voorkomt in de verzameling. Men drukt dit ook wel uit door te zeggen dat in een verzameling geen duplicaten voorkomen. Een andere zaak is hoe wij de elementen van een verzameling beschrijven. Hierbij onderscheidt men beschrijving door *enumeratie* ('opsomming') en beschrijving met behulp van *predicaten* (beweringsvormen). Overigens kan een verzameling ook weergegeven worden met een 'geheel verbale omschrijving', zoals:

Voorbeeld 2.1

A = de verzameling van alle even gehele getallen tussen 7 en 15;
met enumeratie: A = {8,10,14,12};
met predicaten: A = $\{g \mid g \in Z \wedge g \ \underline{\text{mod}}\ 2 = 0 \wedge 7 < g < 15\}$. □

Opmerking. {8,10,14,12} en {8,14,8,10,12} zijn beide andere omschrijvingen van dezelfde bovengenoemde verzameling A, die uit 4 elementen bestaat. Uit bovenstaande moge duidelijk zijn, dat bij de beschrijving van een verzameling de volgorde der elementen er niet toe doet.

Wij spreken van een eindige c.q. niet-eindige verzameling, naar gelang het aantal elementen eindig c.q. niet-eindig is. Zo is de verzameling A van voorbeeld 2.1 eindig en de verzameling N van de natuurlijke getallen niet-eindig.

In dit boek zal verder *uitsluitend* van *eindige verzamelingen* sprake zijn.

Om het aantal elementen van een verzameling V aan te geven, wordt de notatie $|V|$ of #V gebruikt. Zo is $|A| = \#A = 4$ (A in voorbeeld 2.1).

In de 'predicaten-weergave' van V_1 komt de uitdrukking '$g \in Z$' voor. Daarmee wordt bedoeld "g is een element van de verzameling Z". '$g \notin Z$' betekent: g behoort niet tot de verzameling Z.

Het $\in$-teken dient goed te worden onderscheiden van het $\subset$-teken (ook wel het $\subseteq$-teken). Dit laatste, het $\subset$-teken, wordt gebruikt om aan te geven dat een verzameling een *deelverzameling* is van een andere verzameling. V_1 is een deelverzameling van V_2, als elk element van V_1 ook tot V_2 behoort.
Notatie: $V_1 \subset V_2$ of $V_1 \subseteq V_2$.

Men ga zelf na dat de volgende uitspraken juist zijn (A in voorbeeld 2.1):

$\{8,10\} \subset A$

$8 \in A$

$\{8\} \subseteq A$

maar de volgende onjuist:

$8 \subset A$

$\{8\} \in A$

Een deelverzameling V_1, die niet gelijk is aan de 'omvattende' verzameling V_2 noemen wij een *echte* deelverzameling van V_2. Hiervoor wordt soms de notatie $V_1 \subsetneqq V_2$ gebruikt.

De verzameling die geen enkel element bevat, noemen wij de *lege verzameling*.
Notatie $\emptyset$.
De lege verzameling is (bij afspraak) deelverzameling van elke verzameling.

Definitie 2.1 De verzameling van alle deelverzamelingen van een gegeven verzameling V noemen wij de *machtsverzameling* (power set) van V. Notatie: P(V). □

Voorbeeld 2.2

Als gegeven is de verzameling $A = \{8,10,14,12\}$, dan is

$$P(A) = \{\emptyset,\{8\},\{10\},\{12\},\{14\},\{8,10\},\{8,12\},\{8,14\},\{10,12\},\{10,14\},\{12,14\},\{8,10,12\},\{8,10,14\},\{8,12,14\},\{10,12,14\},\{8,10,12,14\}\}.$$

Dus $|P(A)| = 16$ □

Het is eenvoudig in te zien dat uit $|V| = n$ volgt $|P(V)| = 2^n$.

'Rekenen' met verzamelingen is mogelijk met behulp van de operaties verenigen, doorsnede nemen, verschil nemen.

Definitie 2.2 De *vereniging* van de verzamelingen V_1 en V_2 is de verzameling, die bestaat uit precies alle elementen die tot minstens een van beide verzamelingen behoren.
Notatie: $V_1 \cup V_2$. □

Definitie 2.3 De *doorsnede* van de verzamelingen V_1 en V_2 is de verzameling, die bestaat uit precies alle elementen die tot beide verzamelingen behoren.
Notatie: $V_1 \cap V_2$. □

Definitie 2.4 Het *verschil* van de verzamelingen V_1 en V_2 is de verzameling, die bestaat uit precies alle elementen van V_1, die niet tot V_2 behoren.
Notatie: $V_1 \setminus V_2$. □

Voorbeeld 2.3

Gegeven: $A = \{8,10,12,14\}$ en $B = \{x \mid x \in Z \wedge 11 \leq x < 20\}$.
Dan is:

$A \cup B = \{8,10,11,12,13,14,15,16,17,18,19\}$

$A \cap B = \{12,14\}$

$A \setminus B = \{8,10\}$

$B \setminus A = \{11,13,15,16,17,18,19\}$ □

2.2 BEWERINGEN

In voorbeeld 2.1 hebben wij in de verzamelingsdefinitie $\{g \mid g \in Z \wedge g \text{ div } 2 = 0 \wedge 7 < g < 15\}$ gebruik gemaakt van predicaten ofwel *beweringsvormen*. De uitdrukking '$g \in Z \wedge g \text{ div } 2 = 0 \wedge 7 < g < 15$' is een *samengestelde* beweringsvorm, die tot stand is gekomen door de beweringsvormen '$g \in Z$', '$g \text{ div } 2 = 0$' en '$7 < g < 15$' met elkaar te verbinden met behulp van '$\wedge$' (het 'en'-teken).

Een beweringsvorm gaat over in een *bewering* als men aan de variabelen, die er in voorkomen een waarde toekent. Als men aan g de waarde 5 toekent, gaat de beweringsvorm '$g \in Z$' over in de bewering '$5 \in Z$'. Bij de waardetoekenning $g := 1,5$, ontstaat de bewering '$1,5 \in Z$'. De bewering '$5 \in Z$' is waar, de bewering '$1,5 \in Z$' is niet waar. De bewering '$5 \text{ mod } 2 = 0$' is onwaar, want $5 \text{ mod } 2$ is gelijk aan 1. De samengestelde bewering '$5 \in Z$' èn

5 mod 2 = 0' is dus niet waar. Dit laatste is een voor de hand liggende conclusie. Zij is bovendien in overeenstemming met de regels van de (wiskundige) logica. Deze regels maken het ons mogelijk te 'rekenen' met beweringen, dat wil zeggen te bepalen of een samengestelde bewering waar of onwaar is.

In deze paragraaf zullen wij nu nader op deze regels ingaan. Daarbij worden beweringen symbolisch weergegeven met namen als b,b1,b2. Waarden van beweringen worden weergegeven met zogeheten waarheidstabellen.

Definitie 2.5 Als b een bewering is, dan is de *ontkenning* van b (notatie: $\neg$b; not b) de bewering, met waarden volgens tabel 1.

b	$\neg$b
w	o
o	w

w = waar
o = onwaar

Tabel 1 - Waarden van $\neg$b.

Nu volgt de definitie voor de 'en' verbinding.

Definitie 2.6 Als b1 en b2 beweringen zijn, dan is de *conjunctie* van b1 en b2 (notatie: b1 $\wedge$ b2; b1 and b2; b1 en b2) de bewering, met waarden als in kolom 'b1 $\wedge$ b2' van tabel 2.

En voor de 'of' verbinding de volgende

Definitie 2.7 Als b1 en b2 beweringen zijn, dan is de *disjunctie* van b1 en b2 (notatie: b1 $\vee$ b2; b1 or b2; b1 of b2) de bewering, met waarden als in kolom 'b1 $\vee$ b2' van tabel 2.

Tenslotte hebben wij voor de 'als...dan' verbinding

Definitie 2.8 Als b1 en b2 beweringen zijn, dan is de *implicatie* met b1 als *linkerlid* en b2 als *rechterlid* (notatie: b1 $\Rightarrow$ b2) de bewering met waarden als in kolom 'b1 $\Rightarrow$ b2' van tabel 2.

b1	b2	b1 $\wedge$ b2	b1 $\vee$ b2	b1 $\Rightarrow$ b2
w	w	w	w	w
w	o	o	w	o
o	w	o	w	w
o	o	o	o	w

w = waar
o = onwaar

Tabel 2 - Waarden van conjunctie, disjunctie, implicatie.

De waarden van b1 $\wedge$ b2 en van b1 $\vee$ b2 zijn 'voor de hand liggend'. Dit geldt niet voor de implicatie. Zeker niet als men deze zou willen

zien als het weergeven van een verband tussen 'oorzaak' (b1) en 'gevolg' (b2). In het dagelijkse spraakgebruik denkt men bij 'als b1 dan b2' vaak aan een oorzakelijk verband tussen b1 en b2. Dit is uitdrukkelijk niet de bedoeling bij de implicatie in de logica. Met '$b1 \Rightarrow b2$' wil alleen het volgende tot uitdrukking gebracht zijn: "als b1 waar is dan moet ook b2 waar zijn", hetgeen op hetzelfde neerkomt als "als b2 onwaar is, dan moet ook b1 onwaar zijn". Het is inderdaad niet moeilijk om (met behulp van een waarheidstabel) aan te tonen dat $\neg b2 \Rightarrow \neg b1$ equivalent is met $b1 \Rightarrow b2$.

Stel gegeven is het predicaat (de beweringsvorm)

$P(x) = x > 8$ (x geheel getal)

Dan zijn de beweringen P(5) en P(3) onwaar, en
de beweringen P(9), P(17) en P(25) waar.

Stel dat verder gegeven zijn de verzamelingen:

$A = \{5,18,3,34,25,50\}$

$B = \{9,17,25\}$

$C = \{5,3\}$.

Dan zijn de volgende beweringen waar:

B1: er is (minstens) een element $x \in A$, waarvoor P(x) waar is;
B2: voor elk element $x \in B$ is P(x) waar;
B3: er is geen enkel element $x \in C$, waarvoor P(x) waar is;

en de volgende beweringen zijn onwaar:

B4: voor elk element $x \in A$ is P(x) waar;
B5: er is een element $x \in B$, waarvoor P(x) niet waar is.

Bovenstaande beweringen kunnen ook worden weergegeven met zogeheten *kwantoren*. Men onderscheidt:
- de *existentiële kwantor* $\exists$; met de betekenis: "er is een element..."
- de *al-kwantor* $\forall$; met de betekenis: "voor elk element...".

Dus:

B1: $\exists x \in A\ [P(x)]$

B2: $\forall x \in B\ [P(x)]$

B3: $\neg(\exists x \in C\ [P(x)])$

B4: $\forall x \in A\ [P(x)]$

B5: $\exists x \in B\ [\neg P(x)]$

Uiteraard kunnen in een uitdrukking meerdere kwantoren voorkomen, zoals in onderstaande beweringen:

B6: $\exists x \in A\ [\exists y \in B\ [y = x]]$

B7: $\forall x \in B\ [\exists y \in A\ [y = 2x]]$

B8: $\exists y \in A\ [\forall x \in B\ [y = 2x]]$

Men ga zelf na dat B6 en B7 waar zijn en B8 onwaar.

Twee existentiële kwantoren mogen altijd verwisseld worden, zonder dat de waarden van de bewering verandert. Zo is $\exists y \in B\ [\exists x \in A\ [y = x]]$ equivalent met B6. Verwisseling van een al- en een existentiële kwantor leidt echter in het algemeen niet tot een equivalente bewering. Zo is B7 niet equivalent met B8.

2.3 FUNCTIES

Voor de volgende definitie hebben wij het (ongedefinieerde) begrip *geordend paar* nodig. Als a en b elementen zijn dan is (a,b) de notatie voor het geordende paar, bestaande uit a en b in de gegeven volgorde. Wij zeggen dat (a,b) = (c,d) dan en slechts dan als a = c èn b = d.

Van het geordende paar (a,b) heet a de *eerste coördinaat* en b de *tweede coördinaat*. Formele notatie: C1(p) respectievelijk C2(p) is de eerste respectievelijk tweede coördinaat van paar p. Dus C1((a,b)) = a en C2((a,b)) = b.

We zijn nu toe aan de definitie van het welbekende begrip functie. Vooraf zij erop gewezen, dat deze definitie (zoals verderop vrijwel elke definitie in dit boek) gegeven wordt in *niet-formele* en in *formele* vorm. Voor een 'rechtvaardiging' van deze 'dubbele vorm' zij verwezen naar het voorwoord.

De nu volgende definitie van functie is niet erg gebruikelijk, maar voor ons doel wel geschikt (BROCK, 92).

Definitie 2.9
(niet-formeel) De verzameling f is een *functie* als
a. elk element van f een geordend paar is
b. f geen twee verschillende paren bevat waarvan de eerste coördinaten gelijk zijn.
(formeel) De verzameling f is een *functie* $\overset{D}{\equiv}$

$\forall p \in f$ [p is een geordend paar]

$\wedge\ \forall p1,p2 \in f\ [C1(p1) = C2(p2) \Rightarrow p1 = p2]$ □

Opmerking. $\overset{D}{\equiv}$ wil zeggen: "is per definitie gelijk aan".

De verzameling van alle eerste coördinaten van een functie f heet het *domein* van f.
Notatie: dom(f).

De verzameling van alle tweede coördinaten van een functie f wordt het *bereik* van f, ook wel de *range* van f, genoemd.
Notatie: rng(f).

Als (x,y) ∈ f, dan heet y *het beeld van x onder f*.
Om deze 'beeldvorming' te benadrukken, wordt in plaats van (x,y) ∈ f ook gebruik gemaakt van de equivalente notatie: (x, f(x)) ∈ f).

Voorbeeld 2.4

Gegeven:

f1 = {(a,5),(b,7),(c,3),(d,5)}

f2 = {5,7,3,5}

f3 = {(a,{5,7}),(b,7),(c,{3}),(d,5)}

Dan volgt hieruit:

f1 is een functie met dom(f1) = {a,b,c,d} en rng(f1) = {5,7,3} en f1(c) = 3;
f2 is geen functie;
f3 is een functie met dom(f3) = dom(f1) en rng(f3) = {{5,7},7, {3},5} en f3(c) = {3}. □

Dikwijls wordt niet het begrip 'functie' maar alleen het begrip 'functie van A naar B' (ook wel 'afbeelding van A naar B') gedefinieerd, waarbij A en B dan verzamelingen zijn. De Brock (loc.cit) wijst er terecht op dat met definitie 2.5 het niet nodig is (maar wel mogelijk) telkens twee verzamelingen expliciet te noemen, als men over een functie wil spreken. In ons geval, waar wij bij de formele opzet van databases veel gebruik zullen maken van het functiebegrip, biedt de gekozen aanpak beslist voordelen.

Wij zullen nu 'functie van A naar B' definiëren.

Definitie 2.10
(niet-formeel) Als A en B verzamelingen zijn, dan is f een *functie van A naar B* als
- f een functie is èn
- het domein van f gelijk is aan A èn
- het bereik van f een deelverzameling is van B.

(formeel) Als A en B verzamelingen zijn dan:

f is een *functie van A naar B* $\stackrel{D}{\equiv}$
f is een functie èn dom(f) = A èn rng(f) $\subseteq$ B.

Notatie: f: A→B □

Voorbeeld 2.5

(f1 en f3 uit voorbeeld 2.4)

a. f1 is een functie van {a,b,c,d} naar {5,7,3}, maar ook een functie van {a,b,c,d} naar {5,7,3,12,16}.
b. f3 is een functie van {a,b,c,d} naar {5,7,3,{5,7},{3}}. □

Wij zijn nu toe aan een heel belangrijke klasse van functies, namelijk de zogeheten verzamelingsfuncties (Engels: set-functions).

Definitie 2.11
(niet-formeel) Een functie {(a1,W1),(a2,W2),...,(ak,Wk)} is een *verzamelingsfunctie*, als elke Wi een verzameling is.
(formeel)
F is een *verzamelingsfunctie* $\stackrel{D}{\equiv}$
F is een functie èn
∀x ∈ dom(F) [F(x) is een verzameling] □

Voorbeeld 2.6

a. Gegeven: F = {(a,{1,2,3,4}),(b,{x,y,z}),(c,{1,k,x,f})}.

Dan volgt hieruit:

F is een verzamelingsfunctie;
dom(F) = {a,b,c};
rng(F) = {{1,2,3,4},{x,y,z},{1,k,x,f}};
F(b) = {x,y,z}

b. Gegeven: G = {(a,{1}),(b,{x})}; H = {(a,1),(b,x)}.

Dan volgt hieruit:

G is een verzamelingsfunctie;
H is geen verzamelingsfunctie;
dom(G) = dom(H); rng(G) ≠ rng(H);
G(a) = {1}; H(a) = 1

c. ART = {(ano,{1...100}),(anm,charstring 20),
(pr,{10...500}),(gew,{1...300}),(vrd,{0...10000}),
(kl,{rd,w,bl})}

waarbij charstring 20 de verzameling is van alle combinaties van maximaal 20 letters, cijfers of andere toegestane tekens. □

Bij dit laatste geval c mag men uiteraard rustig bij de linkerkolom van onderstaand lijstje 'denken' aan de rechterkolom van dat lijstje. Dit maakt uiteraard het voorbeeld wiskundig niet meer waardevol, maar geeft wel een indicatie hoe grootheden 'uit de werkelijkheid' kunnen worden gerepresenteerd met exact gereedschap.

ART	ARTIKEL
ano	artikelnummer
anm	artikelnaam
pr	prijs
gew	gewicht
vrd	voorraad
kl	kleur

Opmerking. In de database-literatuur wordt het woord 'domein' (Engels: domain) gebruikt om de verzameling aan te geven die het beeld is van een element van het domein van een verzamelingsfunctie. In genoemde literatuur zal ART(ano) het domein van (het attribuut) ano heten. Dit is uiteraard erg verwarrend. Wij zullen het woord domein dan ook uitsluitend gebruiken in de (wiskundige) betekenis van het woord, namelijk verzameling van eerste coördinaten (ofwel af te beelden verzameling).

Tenslotte kunnen wij nu het begrip produkt van een verzamelingsfunctie invoeren.

Definitie 2.12
(niet-formeel) Als F de verzamelingsfunctie {(a1,V1),(a2,V2),..., (ak,Vk)} is, dan is het *produkt van F* gelijk aan de verzameling die bestaat uit alle functies f waarvoor geldt:

f = {(a1,v1),(a2,v2),...,(ak,vk)} met

vi element van Vi $(1 \leq i \leq k)$

(formeel) Als F een verzamelingsfunctie is, dan is

het *produkt van F* $\stackrel{D}{\equiv}$

{f | f is een functie èn dom(f) = dom(F) èn

$\forall a \in$ dom(f) [f(a) $\in$ F(a)] }.

Notatie: Π(F) □

Het woord 'produkt' is hier geschikt gekozen, omdat het aantal elementen van Π(F), dus |Π(F)| gelijk is aan het produkt van |V1| * |V2| * ... * |Vk|. Er geldt dan ook voor elke (niet-lege) verzamelingsfunctie F:

Stelling 2.1 (Π(F) = ∅) dan en slechts dan als ($\exists a \in$ dom(F) [F(a) = ∅]).

In woorden: als het produkt van een verzamelingsfunctie
$F = \{(a1,V1),(a2,V2),...,(a3,V3)$ leeg is, dan is er een Vi leeg en omgekeerd.

Een element van het produkt $\Pi(F)$ heet een *tupel* (Engels: tuple) van $\Pi(F)$.

Voorbeeld 2.7

a. $\Pi(F)$ van F uit voorbeeld 2.6a bestaat uit $4 \times 3 \times 4 = 48$ elementen. Een tupel van $\Pi(F)$ is bijvoorbeeld

$f = \{(a,1),(b,y),(c,x)\}.$

b. $\Pi(G)$ van G uit voorbeeld 2.6b bestaat uit precies één element en wel het element

$\{(a,1),(b,x)\}.$

Let wel: $\Pi(G) = \{\{(a,1),(b,x)\}\}.$

c. Onderstaand is een deelverzameling van $\Pi(F)$ van F uit voorbeeld 2.2a weergegeven:

$$Y = \{\{a,1),(b,y),(c,x)\},\{(a,1),(b,z),(c,x)\},\{(a,4),(b,x),(c,x)\},\{(a,3),(b,z),(c,k)\},\{(a,1),(b,z),(c,1)\}\} \subset \Pi(F).$$

Zo'n deelverzameling van een produktverzameling kan eenvoudig worden weergegeven met behulp van een tabel. Zo is de tabelweergave van Y als volgt:

a	b	c
1	y	x
1	z	x
4	x	x
3	z	k
1	z	1

□

Wil men een functie 'beperken' tot een deel van het domein, dan is de volgende definitie van belang.

Definitie 2.13
(niet-formeel) Als f een functie is en X is een deelverzameling van het domein van f, dan is de *restrictie van f tot X* de functie, die bestaat uit de paren (x,y) van f, waarbij x een element is van X.
(formeel) Als f een functie is en $X \subseteq dom(f)$, dan is

de restrictie van f tot X $\overset{D}{\equiv} \{(x,y) \mid (x,y) \in f \wedge x \in X\}.$

Notatie: $f/\overline{X}$ □

En voor een soortgelijke 'beperking' van een deelverzameling van het produkt van een verzamelingsfunctie:

Definitie 2.14

(niet-formeel) Als F een verzamelingsfunctie is en X een deelverzameling van het domein van F en Y een deelverzameling van het produkt van F, dan is de *projectie van Y op X* de verzameling van de tot X gerestricteerde functies van Y.

(formeel) Als F een verzamelingsfunctie is èn $X \subset \mathrm{dom}(F)$ èn $Y \subseteq \Pi(F)$, dan is

$$\textit{de projectie van Y op X} \overset{D}{\equiv} \{f\lceil X \mid f \in Y\}.$$

Notatie: $Y//\overline{X}$. □

Voorbeeld 2.8

Bij f respectievelijk Y van voorbeeld 2.7a respectievelijk c:

$f/\overline{\{a,b\}} = \{(a,1),(b,y)\};$

$f/\overline{\{a\}} = \{(a,1)\};$

$Y//\overline{\{a,b\}} = \{\{(a,1),(b,y)\},\{(a,1),(b,z)\},\{(a,4),(b,x)\},$
$\{(a,3)(b,z)\}\}.$

$Y//\overline{\{a\}} \quad \{\{(a,1)\},\{(a,4)\},\{(a,3)\}\}.$

In tabelvorm luidt ditzelfde voorbeeld voor $Y\mid\lceil\{a,b\}$ en $Y\mid\lceil\{a\}$:

$Y//\overline{\{a,b\}} =$

a	b
1	y
1	z
4	x
3	z

$Y//\overline{\{a\}} =$

a
1
4
3

Uit de bovengenoemde definities volgt direct, ten aanzien van de restrictie:

Stelling 2.2 Als f een functie is en $X \subset dom(f)$, dan

a. $f/\overline{X}$ is een functie en $f/\overline{X} \subseteq f$.

b. $dom(f/\overline{X}) = X$ èn $\forall x \in X[f/\overline{X}(x) = f(x)]$.

c. $f/\overline{\emptyset} = \emptyset$. □

Het zal in het vervolg uiterst nuttig blijken, te kunnen beschikken over het volgende begrip.

Definitie 2.15
(niet-formeel) Een verzamelingsfunctie F = {(a1,V1),(a2,V2),..., (ak,Vk)} is *regulier* als elke Vi niet leeg is.
(formeel) Als F een verzamelingsfunctie is, dan is

F is een *reguliere verzamelingsfunctie* $\stackrel{D}{\equiv}$

$F \neq \emptyset$ èn $\forall a \in dom(F)\ [F(a) \neq \emptyset]$. □

Een reguliere verzamelingsfunctie heeft de volgende 'geschikte' eigenschap:

Stelling 2.3 Als F een reguliere verzamelingsfunctie is en $X \subset dom(F)$, dan geldt:

$(\Pi(F))//\overline{X} = \Pi(F/\overline{x})$. □

In 'woorden' luidt deze stelling: de projectie van het produkt van een reguliere verzamelingsfunctie F op X is gelijk aan het produkt van de restrictie van F tot X. Het doet er dus niet toe of men eerst de verzamelingsfunctie 'beperkt' en dan het produkt neemt of omgekeerd, de 'beperking' pas tot stand brengt nadat het produkt gevormd is.

Nogmaals zij opgemerkt dat dit 'goed gaat', omdat de voorwaarde is gesteld dat F regulier is. (Voor de liefhebbers: dit is wel een voldoende, maar geen nodige voorwaarde voor stelling 2.3.) Maar deze regulariteit is nu juist hetgeen overeenkomt met ons intuïtieve beeld van de attributen, die wij willen representeren. Immers, het is niet voorstelbaar, dat wij attributen zouden willen representeren, waar geen enkele waarde aan kan worden toegekend. Eenwaardige attributen kunnen wij ons wel voorstellen en zijn dan ook in het formele model toegelaten. Overigens zullen wij van eenwaardige attributen geen gebruik maken, omdat er geen enkele informatie aan kan worden ontleend. In feite zullen wij dus alleen te maken hebben met attributen, waaraan minstens twee verschillende waarden kunnen worden toegekend.

OPGAVEN

2.1 Gegeven zijn de verzamelingen A,B,C,D,E.
$A = \{1,2,3,4,5\}$; $B = \{2,4,6,8\}$; $C = \{3,4,7,9\}$;
$D = \{7,8,10,13\}$; $E = \emptyset$.

a. Geef de elementen van:

$V_1 = A \cup B$	$V_5 = (A \cap C) \cup (B \cap C)$	$V_9 = D \cup E$
$V_2 = A \cap B$	$V_6 = (A \cup B) \cup C$	$V_{10} = D \setminus E$
$V_3 = A \cap D$	$V_7 = A \cup (B \cup C)$	$V_{11} = E \setminus D$
$V_4 = (A \cup B) \cap C$	$V_8 = D \cap E$	$V_{12} = (A \setminus B) \setminus C$

b. Bepaal:

$|A \cup B|$; $|A| + |B|$; $|E|$; $|D \setminus E|$.

c. Welke van de volgende beweringen zijn waar?

c1. $\{2\} \in B$

c2. $7 \in A \cup C$

c3. $\{\{2\}\} \subset B$

c4. $(A \cap B) \cup D = (A \cup D) \cap (B \cup D)$

c5. $(A \setminus B) \setminus D = A \setminus (B \setminus D)$

c6. $E \subset A$

c7. $E \in A$

c8. $B \cap C \in A$

c9. $\{t \mid t \in N \wedge t > 0 \wedge t \underline{\text{ mod }} 2 = 0 \wedge t \leq 9\} \subset B$

c10. $\{t \mid t \in N \wedge t > 0 \wedge t \underline{\text{ mod }} 2 = 0 \wedge t \leq 9\} \supset B$

c11. $\{t \mid t \in N \wedge t > 0 \wedge (t \underline{\text{ mod }} 2 = 1 \vee t \leq 9)\} \supset B$

c12. $\exists x \in A$ [x is even]

c13. $\forall x \in A\ [\exists y \in B\ [y>x]]$

c14. $\exists y \in B\ [\forall x \in A\ [y>x]]$

d. Beschrijf alle elementen van $P(B)$ en van $P(B \cap C)$.

e. Toon, dat als $|V| = n$, geldt: $|P(V)| = 2^n$.

f. Welke van de volgende beweringen zijn waar?

f1. $\{1\} \subset P(A)$

f2. $\{\{7\},\{7,8\}\} \subset P(D)$

f3. $P(B \cap C) = P(B) \cap P(C)$

f4. $\{E\} \in P(A)$

2.2 a. Van functie f is gegeven:

$dom(f) = \{a,b,c\}$;

$f(a) = 3$; $f(b) = f(c) = 3\ f(a)$

Geef een beschrijving van f (enumeratie).

b. Van functie g is gegeven:

$dom(g) = \{p,q,r,s\}$;

$\forall x \in dom(g)\ [g(x) = x]$

Geef een beschrijving van g (enumeratie).

c. Gegeven h: $x \rightarrow x^2 + 3 \mid x \in \{3,5,7\}$.

Geef een beschrijving van h (enumeratie).

2.3 Ga de juistheid van stelling 2.2 na.

2.4 Gegeven zijn de volgende zeven functies:

F1 = {(kleur,{fout,goed}),(produkt,{fout,goed})}.

F2 = {(mnr,{1,2,3}),(nm,{tf,st}),(kl,{w,rd})}.

F3 = {(a,1),(b,{1,2,3}),(c,{4,5}),(d,{2,1,3})}.

F4 = {(gew,{5}),(nm,{a,b}),(kl,{w,zw}),(pr,{10,15,20})}.

F5 = {(gew,{0}),(nm,{a,b}),(kl,{w,zw}),(pr,{10,15,20})}.

F6 = {(gew,∅),(nm,{a,b}),(kl,{w,zw}),(pr,{10,15,20})}.

F7 = {(a,{5,7}),(b,{p,q,r}),(c,{{p,q},{q,r},{p,r}})}.

a. Geef: dom(F5); rng(F5); dom(F1); rng(F1); dom(F7); rng(F7).

b. Geef: F3(a); F2(nm); F5(gew); F6(gew).

c. Vul in (indien mogelijk):

F2(...) = {tf,st}; F5(...) = {15,10,20}; F1(...) = fout;

F7(...) = {p,q}; F3(...) = 1; F5(...) = 0;

F3(...) = {1,2,3}.

d. Bepaal: $F1/\overline{\{kleur\}}$;

$F4/\overline{\{gew,kl\}}$; $F4/\overline{\{kl,nm\}}$;

$F5/\overline{\{gew,kl\}}$; $F5/\overline{\{kl,nm\}}$.

e. Ga voor elke functie na of het een verzamelingsfunctie is. Zo ja, geef het aantal elementen van het produkt van de desbetreffende verzamelingsfunctie en geef (zo mogelijk) twee elementen van dit produkt.

f. Bepaal: $\Pi(F4)/\!/\overline{\{gew,kl\}}$; $\Pi(F4\ /\overline{\{gew,kl\}}$.

$\Pi(F6)/\!/\overline{\{gew,kl\}}$; $\Pi(F6/\overline{\{gew,kl\}}$.

$\Pi(F4)/\!/\overline{\{kl,pr\}}$; $\Pi(F4/\overline{\{kl,pr\}}$.

$\Pi(F6)/\!/\overline{\{kl,pr\}}$; $\Pi(F6/\overline{\{kl,pr\}}$.

g. $t \in \Pi(F2)$

Welke van onderstaande beweringen is waar?

g1. $t(mnr) = t/\overline{\{mnr\}}$

g2. $t(mnr,nm) = t/\overline{\{mnr,nm\}}$

g3. $(m,t(mnr)) = t/\overline{\{mnr\}}$.

3 TUPELTYPE EN RECORDTYPE; TABELTYPE EN FILETYPE

3.1 TUPELTYPE EN RECORDTYPE

Elk individueel object, dat actueel van belang is, dus elke actuele objectoccurrence, zullen wij de in database gepresenteerd willen zien. Deze representatie zal dan de vorm hebben van een verzameling waarden van relevante attributen. Zo zullen bijvoorbeeld van een patiënt in de database liggen opgeslagen de waarden van de attributen: pnr (patiëntnummer), nm (naam), adr (adres), wpl (woonplaats), gbd (geboortedatum), gsl (geslacht). Het is gebruikelijk zo'n verzameling bij elkaar behorende waarden een *record* te noemen. Bij een object kunnen vele mogelijke actuele objectoccurrences horen. In programmeertalen als Pascal wordt dan ook het zogeheten recordtype gebruikt. Zo'n recordtype is de verzameling van alle mogelijke records, dus van alle mogelijke combinaties van attribuutwaarden. Wij zullen zien dat recordtypen in feite heel bijzondere gevallen zijn van zogeheten tupeltypen.

Een tupeltype is de verzameling van alle mogelijks tupels. En een tupel is dan de representatie van een object-occurrence. Zo kunnen de gegevens van een bepaalde patiënt als volgt met een tupel t1 (een functie) worden weergegeven:

t1 = {(pnr,15),(nm,p.jansen),(adr.rondweg 10),(wpl,eindhoven), (gbd,250116),(gsl,m)}.

En van een andere patiënt zal onderstaand tupel t2 de actuele toestand weergeven:

t2 = {(pnr,27),(nm,a.brakes),(adr,woertlaan 15),(wpl,nuenen), (gbd,350310),(gsl,v)}.

Een en ander leggen wij nu vast in de volgende definitie.

Definitie 3.1
(niet-formeel) Een *tupeltype* is een niet-lege verzameling functies, waarvoor geldt, dat zij allen hetzelfde domein hebben.

(formeel) De verzameling T is een *tupeltype* $\overset{D}{\equiv}$

$T \neq \emptyset$ èn

$\forall f \in T$ [f is een functie $\wedge f \neq \emptyset$] èn

$\forall f1, f2 \in T$ [dom(f1) = dom(f2)]. □

Zo is Y van voorbeeld 2.7c een tupeltype. Uit dit voorbeeld is ook duidelijk, dat een tupeltype eenvoudig in tabelvorm kan worden weergegeven. Het gemeenschappelijke domein fungeert dan als tabelhoofd. Vanwege de gemeenschappelijkheid van het domein, zullen wij ook spreken van het *domein van een tupeltype*, en daarmee bedoelen het gemeenschappelijke domein van de functies van dit tupeltype. Een element van het domein van een tupeltype wordt een *attribuut* van dit tupeltype genoemd. Het genoemde voorbeeld Y is, zoals we zagen in voorbeeld 2.7c, een deelverzameling van het produkt van een verzamelingsfunctie. Dit is niet toevallig. Het is namelijk eenvoudig aan te tonen, dat bij elk tupeltype een verzamelingsfunctie kan worden gedefinieerd zodanig dat het onderhavige tupeltype een deelverzameling is van het produkt van genoemde verzamelingsfunctie. Een element van een tupeltype is dus een tupel.

Voor het genoemde voorbeeld Y kan F van voorbeeld 2.6a als 'bijbehorende' verzamelingsfunctie fungeren, maar ook

$F' = \{(a,\{1,3,4\}),(b,\{x,y,z\}),(c,\{1,k,x\},\}.$

Het verschil tussen F en F' zit uiteraard niet in het domein, maar in de beelden van het domein (dus de waardenverzameling behorende bij elke attribuut).

Verder kan nog worden opgemerkt dat F' de *'kleinste'* bij Y behorende verzamelingsfunctie is, waarmee wij bedoelen dat F' de kleinst mogelijke waardenverzameling per attribuut bevat zodanig dat $Y \subset \Pi(F')$. Het is niet moeilijk in te zien dat bij elk tupeltype X een kleinste verzamelingsfunctie FX bestaat.

Aangezien een tupeltype wordt gebruikt als de representatie van de verzameling mogelijke objectoccurrences, ligt de volgende naamgeving voor de hand. Een bij een tupeltype behorende verzamelingsfunctie (in zojuist geschetste zin) wordt ook wel een *objectkarakterisering* genoemd. Uiteraard kunnen wij dan verder ook spreken van de *kleinste objectkarakterisering* van een tupeltype.

Beschouwen we nogmaals het reeds eerder genoemde voorbeeld Y van een tupeltype en de daarbij behorende objectkarakterisering F'. We moeten dan vaststellen, dat ofschoon F' de kleinste bijbehorende objectkarakterisering is, Y toch niet gelijk is aan het gehele produkt $\Pi(F')$. Met andere woorden: $\Pi(F')$ bevat tupels, die niet in Y voorkomen. Men kan het ook zo stellen: er zijn blijkbaar tupels in $\Pi(F')$ die niet als representanten van 'bijbehorende' objectoccurrences kunnen optreden, om de eenvoudige reden dat deze objectoccurrences

niet mogelijk zijn.

Hoe kunnen nu tupels worden uitgesloten? Door ze allemaal apart te noemen. Maar deze methode zal over het algemeen onwerkbaar zijn, omdat het gaat om zeer grote aantallen. Hiervoor hebben wij dan ook de aanpak met behulp van de zogeheten *constraints*, ook wel genoemd de *integrity-constraints*.

Nu kunnen wij de constraints nog in diverse klassen onderscheiden. Deze klassen komen successievelijk aan bod in dit en de volgende hoofdstukken. Wij beginnen nu met de zogeheten tupelconstraint.

Een tupelconstraint geeft weer waaraan de combinatie van attribuutwaarden van een tupel moet voldoen. Een *tupelconstraint* is dus een predicaat over het produkt van een objectkarakterisering.

Ter verduidelijking het volgende voorbeeld.

Voorbeeld 3.1

Gegeven zijn de verzamelingsfuncties F en ART (van voorbeeld 2.6a en 2.6c):

F = {(a,{1,2,3,4}),(b,{x,y,z}),(c,{1,k,x,f})}.

ART = {(ano,{1...100}),(anm,charstring 20),(pr,{10...500}),
(gew,{1...300}),(vrd,{0...10000}),(kl,{rd,w,bl})}.

De tupelconstraints C1, C2 en C3 worden nu als volgt gedefinieerd.

C1(t) := [(t(a) = 1) ⇒ t(b) ≠ y ∧ t(c) = k] over de verzameling Π(F);

C2(t) := [(t(ano) < 75) ⇒ t(gew) > 20] over de verzameling Π(ART);

C3(t) := [t(pr) × t(vrd) < 100000] over de verzameling Π(ART).

□

Een tupel t van Π(F) zal voldoen aan de tupelconstraint C1, als, in geval attribuut a de waarde 1 heeft, attribuut b een waarde ongelijk y heeft en attribuut c de waarde k. De tupels {(a,1),(b,x),(c,k)} en {(a,2),(b,y),(c,1)} voldoen dus aan C1, maar tupel {(a,1),(b,y),(c,k)} niet.

Door C1 worden 10 tupels van Π(F) 'uitgeschakeld' (men ga dit na!).

Stel constraint C4 is als volgt gedefinieerd:

C4(t) := [t(a) ≠ 2] over Π(F), F als in voorbeeld 3.1.

Het merkwaardige aan C4 is, dat deze constraint overbodig is, als in plaats van F(a) = {1,2,3,4} zou worden gesteld F(a) = {1,3,4}. Met

andere woorden: C4 is eigenlijk geen constraint op tupelniveau, maar op een niveau 'lager', op attribuutniveau. C4 zou men dan ook gevoeglijk een attribuutconstraint kunnen noemen. Meer algemeen: een *attribuutconstraint* geeft aan waaraan de waarde van een attribuut (los van andere attributen) moet voldoen.

Wij spreken nu van een *echte tupelconstraint* als deze geen attribuutconstraint bevat. C4 is een duidelijk voorbeeld van een niet-echte tupelconstraint. Niet-echte tupelconstraints kunnen echter ook in meer 'verhulde' vorm voorkomen, zoals constraint C5, die als volgt is gedefinieerd:

$C5 := [(t(a) \in t\{1,2\} \Rightarrow t(b) = x) \wedge (t(a) \in t\{3,4\} \Rightarrow t(b) \neq y)]$.

De tupelconstraint C5 bevat de attribuutconstraint C6 en de echte tupelconstraint C7, waarbij

$C6 := [t(b) \neq y]$ (dus $y \notin F(b)$) en

$C7 := [t(a) \in t\{1,2\} \Rightarrow t(b) = x]$.

(C5, C6 en C7 constraints over eerder genoemde $\Pi(F)$.)

Aangezien C2 en C3 betrekking hebben op dezelfde verzameling, kan men ook spreken van de conjunctie $C2 \wedge C3$ als één constraint. Door van eenzelfde produkt de conjunctie te nemen van alle constraints over dat produkt, kan men in feite spreken over een *karakteriserend* tupelconstraint over dat produkt voor het gegeven tupeltype. Gegeven een tupeltype T ligt dus vast: de bijbehorende kleinste objectkarakterisering K en een bijbehorende karakteriserende tupelconstraint TC, zodanig dat

(A) $\{t \mid t \in \Pi(K) \wedge TC(t)\} = T$.

Hierbij moet nog wel worden opgemerkt, dat TC op equivalentie na vastligt. Hiermee bedoelen wij dat TC vervangen kan worden door TC', mits de uitdrukking sub A waar blijft: met andere woorden $TC(t) \Leftrightarrow TC'(t)$ voor alle $t \in \Pi(K)$.

Voorbeeld 3.2

We zagen al dat bij Y van voorbeeld 2.7c als kleinste objectkarakterisering hoort

$F' = \{(a,\{1,3,4\}),(b,\{x,y,z\}),(c,\{1,k,x\})\}$.

Een karakteriserende tupelconstraint over $\Pi(F')$ voor Y is bijvoorbeeld:

$$TC1(t) = [(t(b) = t(c) \Leftrightarrow t(a) > 3) \wedge (t(c) = k \Leftrightarrow t(a) = 3) \wedge (t(b) \neq z \Rightarrow t(c) = x)]$$

(voor nadere toelichting zie onderstaande tabel)

maar ook de daarmee equivalente

$$\begin{aligned} TC2(t) = \; & [(t(a) = 1 \Rightarrow (t(b) \neq x \wedge t(c) \neq k)) \wedge \\ & (t(b) = y \Rightarrow t(c) \neq 1) \wedge \\ & (t(a) = 3 \Rightarrow t(b) \notin \{x,y\} \wedge t(c) \notin \{1,x\}) \wedge \\ & (t(a) = 4 \Rightarrow t(b) \notin \{z,y\} \wedge t(c) \notin \{1,k\})]. \end{aligned}$$

Met behulp van onderstaande tabel, waarin alle tupels van $\Pi(F')$, moge duidelijk zijn dat inderdaad geldt: $Y = \{t \mid t \in \Pi(F') \wedge TC1(t)\}$. Hierbij is

$C1(t) := [t(b) = t(c) \Leftrightarrow t(a) > 3]$

$C2(t) := [t(c) = k \Leftrightarrow t(a) = 3]$

$C3(t) := [t(b) \neq z \Rightarrow t(c) = x]$.

Een streepje in een C-kolom wil zeggen, dat het tupel niet voldoet aan de desbetreffende constraint.

De lezer ga zelf de juistheid van het gestelde na met TC2.

a	b	c	C1	C2	C3	a	b	c	C1	C2	C3	a	b	c	C1	C2	C3
1	x	1			-	3	x	1		-	-	4	x	1	-		-
1	x	k		-	-	3	x	k			-	4	x	k	-	-	-
1	x	x	-			3	x	x	-	-		**4**	**x**	**x**			
1	y	1			-	3	y	1		-	-	4	y	1	-		-
1	y	k		-	-	3	y	k			-	4	y	k	-	-	-
1	**y**	**x**				3	y	x		-		4	y	x	-		
1	**z**	**1**				3	z	1		-		4	z	1	-		
1	z	k		-		**3**	**z**	**k**				4	z	k	-	-	
1	**z**	**x**				3	z	x		-		4	z	x	-		

□

In het algemeen zal en kan het niet zo zijn dat vanuit een gegeven tupeltype T de bijbehorende kleinste objectkarakterisering en een bijbehorende karakteriserende tupelconstraint wordt 'gezocht'. Integendeel: uitgangspunt zal zijn een objectkarakterisering (attributen met bijbehorende waardenverzamelingen), waar bovenop dan de tupelconstraint(s) worden gedefinieerd. Hiermee ligt dan uiteraard een tupeltype vast.

Voorbeeld 3.3

Gegeven de objectkarakterisering

$K = \{(a,\{1,2,3,4\}),(b,\{x,y,z\}),(c,\{1,k,x,f\})\}$ (voorbeeld 2.6a)

èn de constraints C1 en C2 over $\Pi(K)$:

$C1(t) = [(t(a) = 1) \Rightarrow (t(b) \neq y \wedge (t(c) = x \vee t(c) = 1))]$

$C2(t) = [(t(a) \neq 1) \Rightarrow (t(b) = y \wedge t(c) \neq x)]$

dan ligt hiermee vast het tupeltype T, onderstaand weergegeven in tabelvorm.

a	b	c
1	x	x
1	x	1
1	z	x
1	z	1
2	y	1
2	y	k
2	y	f
3	y	1
3	y	k
3	y	f
4	y	1
4	y	k
4	y	f

□

Voorbeeld 3.4

De objectkarakterisering ART en de constraints C2 en C3 van voorbeeld 3.1.
Het tupeltype T-ART waarvoor geldt

$T\text{-}ART = \{t \mid t \in \Pi(ART) \wedge (C2 \wedge C3)(t)\}$

bevat nu zeer vele tupels. Het is uiteraard praktisch onmogelijk T-ART in tabelvorm weer te geven. □

Het is theoretisch denkbaar dat een tupeltype gelijk is aan het produkt van de bijbehorende kleinste objectkarakterisering, dus een tupeltype zonder enige tupelconstraints (formeel: TC = true). En dit

is dan gelijk aan het record-type in (de programmeertaal) Pascal. Het Pascal record-type is dus wel een heel bijzonder geval van het tupeltype.

Nu wij kunnen beschikken over waardenverzamelingen van het tupeltype, kunnen wij variabelen declareren van een tupeltype.

Voorbeeld 3.5

Met:

```
var FUN1,FUN2 : T;
    A : T-ART
endvar;
```

worden variabelen gedeclareerd van het type T (voorbeeld 3.3) respectievelijk T-ART (voorbeeld 3.4). Met

```
FUN1.a := 3; FUN1.b := y; FUN1.c := x
```

wordt dan bewerkstelligt dat de variabele FUN1 een bepaalde waarde krijgt. Met FUN2 := FUN1 krijgt FUN2 dezelfde waarde.

In dit voorbeeld wordt uitgegaan van een syntax à la Pascal. De componenten van de variabelen zijn de attributen van de bijbehorende tupeltypen. □

3.2 TABELTYPE EN FILETYPE

Van een bepaald object is meestal niet precies één objectoccurrence actueel, maar is sprake van meerdere actuele objectoccurrences, bijvoorbeeld meerdere artikelen. De representatie van een verzameling V1 actuele occurrences van eenzelfde object kan uiteraard geschieden in de vorm van een deelverzameling van een tupeltype. Voor de representatie van een andere verzameling V2 actuele occurrences van hetzelfde object op een ander moment kan dan een andere deelverzameling worden genomen van hetzelfde tupeltype.

Voorbeeld 3.6

Onderstaand worden in de vorm van twee tabellen verschillende deelverzamelingen D1 en D2 van het tupeltype T-ART (voorbeeld 3.4) gegeven.

D1 =

ano	anm	pr	gew	vrd	kl
5	stoel	25	21	153	rd
7	stoel	27	29	61	rd
93	tafel	132	42	18	w
78	bank	98	38	12	w

D2 =

ano	anm	pr	gew	vrd	kl
17	kast	275	112	18	rd
7	stoel	27	29	61	rd
5	stoel	25	21	153	rd
93	tafel	132	42	23	w
78	bank	98	38	17	w

□

D1 en D2 in bovenstaand voorbeeld kunnen ook worden gezien als deelverzamelingen van twee verschillende tupeltypen, mits de domeinen (attributenverzamelingen) van beide gelijk zijn. Aangezien een niet-lege deelverzameling van een tupeltype per definitie weer een tupeltype is, kunnen D1 en D2 zelfs gezien worden als twee verschillende tupeltypen met uiteraard eenzelfde domein. Twee tupeltypen met eenzelfde domein noemen wij *domeinverwante tupeltypen* of kortweg *verwante tupeltypen*. Aangezien een tabel een niet-formele weergave is van een tupeltype, kan op niet-formele wijze gesproken worden van *verwante tabellen*, als zij hetzelfde tabelhoofd hebben.

De gebruikelijke benaming voor een (deelverzameling van een) tupeltype als representatie van de verzameling actuele occurrences van een object is file (bestand). De verzameling actuele occurrences van een object kan (in de loop der tijd) erg wisselend zijn. Analoog aan het eerder genoemde recordtype wordt in programmeertalen als Pascal dan ook het zogeheten filetype gebruikt. Wij zullen ook hier weer zien, dat filetypen heel bijzondere gevallen zijn, en wel van zogeheten tabeltypen.

Definitie 3.2
(niet-formeel) Een *tabeltype* is een verzameling verwante tabellen of een verzameling verwante tabellen verenigd met de verzameling, die bestaat uit de lege verzameling.
(formeel) De verzameling TT is een *tabeltype* $\overset{D}{\equiv}$

$$TT = V \quad \text{òf} \quad TT = \{\emptyset\} \cup V,$$

waar V een verzameling verwante tupeltypen is. □

Een element van een tabeltype heet een *tabel*. Om dit formele begrip 'tabel' te onderscheiden van het eerder gebruikte niet-formele begrip 'tabel' (eerste maal in voorbeeld 2.7), zou men voor dit laatste steeds een andere benaming, bijvoorbeeld 'tabelfiguur', kunnen gebruiken. In dit boek wordt echter geen gebruik gemaakt van zo'n aparte benaming, omdat het nogal gekunsteld is en uit de context wel duidelijk is of sprake is van het formele dan wel het niet-formele begrip.

Een tabel is de formele representatie van de verzameling actuele occurrences van een object, dus de representatie van een bestand op een bepaald moment. Als de lege verzameling tot een tabeltype behoort, betekent dit, dat het te representeren bestand leeg kan zijn. Een leeg bestand hoeft uiteraard niet tot de toegestande mogelijkheden te behoren.

Voorbeeld 3.7

Onderstaand worden drie voorbeelden van tabeltypen gegeven, TT1, TT2, TT3.

1. $TT1 = \{\emptyset, \Pi(F), Y, T\}$, waarbij F,Y,T in de voorbeelden 2.6a, 2.7c en 3.3 voorkomen.
2. $TT2 = TT1 \setminus \{\Pi(F)\}$. (Let wel: $TT2 = \{\emptyset, Y, T\}$ en $TT1 \setminus \Pi(F) = TT1$)
3. $TT3 = TT1 \cup \{\{\{(a,5),(b,y),(c,x)\},$
$\{(a,4),(b,z),(c,f)\}\},$
$\{\{(a,5),(b,z),(c,f)\},$
$\{(a,4),(b,z),(c,f)\},$
$\{(a,3),(b,y),(c,1)\}\}\}.$ □

Aangezien verwante tupeltypen hetzelfde domein, dus dezelfde attributen hebben, spreken wij van het *domein*, de *attributen van een tabeltype* en bedoelen dan daarmee het gemeenschappelijke domein van de verwante tupeltypen van dit tabeltype.

Bij elk tabeltype TT kan (minstens) één tupeltype T worden gedefinieerd zodanig dat elk van de verwante tupeltypen van TT een deelverzameling is van T. Zo'n tupeltype T noemen wij dan een *karakteriserend* tupeltype voor TT.

Voorbeeld 3.8

Onderstaand worden enkele voorbeelden van karakteriserende tupeltypen gegeven.

Tabeltype (zie vb. 3.7)	Karakteriserende tupeltypen
a. TT1	$\Pi(F)$ (vb. 2.7a)
b. TT2	b.1. $\Pi(F)$ (vb. 2.7a)
	b.2. T (vb. 3.3) $\cup$ Y (vb. 2.7c)
c. TT3	c.1. $\Pi(F)$ (vb. 2.7a) $\cup$ $\{\{(a,5),(b,y),(c,x)\}, \{(a,5),(b.z),(c.f)\}\}$.
	c.2. $\Pi(G)$, waarbij $G = \{(a,\{1,2,3,4,5\}),(b,\{x,y,z\}), (c,\{1,k,x,f\})\}$.

□

Twee verschillende karakteriserende tupeltypen van eenzelfde tabeltype hebben uiteraard hetzelfde domein. Als TT een tabeltype is, noemen wij de verzameling tupels

$$MT = \{t \mid \exists T \in TT[t \in T]\}$$

het *kleinste* karakteriserende tupeltype van TT. Merk op dat MT inderdaad een karakteriserend tupeltype van TT is. Er geldt dus voor elk karakteriserend tupeltype KT van TT: KT $\supset$MT.

Zo is T $\cup$ Y het kleinste karakteriserende tupeltype van tabeltype TT2 (vb. 3.8b).

In het algemeen zal niet elke deelverzameling van het kleinste karakteriserende tupeltype MT van een tabeltype TT tot dit tabeltype behoren.

Voorbeeld 3.9

Gegeven is tabeltype TT4 = $\{\emptyset$,E1,E2,E3,E4$\}$ met Ei (in tabelvorm):

E1 =

a	b	c
1	y	k
2	y	1
3	x	x

E3 =

a	b	c
1	y	k
3	x	x

E2 =

a	b	c
4	2	f
2	y	f

E4 =

a	b	c
4	2	1
1	x	k
3	x	x

Dan is het kleinste karakteriserende tupeltype MT van TT4 (in tabelvorm):

MT =

a	b	c
1	x	k
1	y	k
2	y	1
2	y	f
3	x	x
4	2	1
4	2	f

Er zijn dus in totaal $2^7 = 128$ deelverzamelingen van MT. Van deze 128 behoren er slechts vijf tot het tabeltype TT4. □

Hoe kunnen nu deelverzamelingen van MT (dus elementen van P(MT)) worden uitgesloten? Evenals bij uitsluiting van tupels bij de definitie van een tupeltype, moet ook hier gezegd worden dat uitsluiting van deelverzamelingen via de weg van enumeratie een 'onbegonnen zaak' is. Wij zullen het ook hier weer doen met behulp van integrity constraints en wel met de klasse van de zogeheten tabelconstraints.

Een tabelconstraint geeft weer waaraan de combinatie van tupels van een tabel moet voldoen. Een *tabelconstraint* is dus een predicaat over de machtsverzameling van een tupeltype.

Voorbeeld 3.10

De constraints C1, C2:

C1(D) := $[\#(D) \leq 5]$ over P(T) (T in vb. 3.3); dus $D \in P(T)$.

C2(D) := $[\forall t1,t2 \in D[t_1(a) = t_2(a) \Rightarrow t_1 = t_2]]$ over P(MT)

(MT in vb. 3.9); dus $D \in P(MT)$.

P(T) bevat $2^{13} = 8192$ deelverzamelingen. Door constraint C1 worden hiervan 5812 'uitgeschakeld'. □

Evenals bij tupelconstraints maken wij bij tabelconstraints onderscheid tussen *echte* en *niet-echte tabelconstraints*. Een echte tabelconstraint is een constraint die geen constraint van 'lagere orde', dus een tupel- of attribuutconstraint, bevat. Stel C3 is als volgt gedefinieerd: C3(D) := $[\forall t \in D[t(a) = 2 \Rightarrow t(c) = f]$ over P(MT)

(MT in vb. 3.9), dan is C3 geen echte tabelconstraint, maar een echte tupelconstraint.

Analoog aan de karakteriserende tupelconstraints bij een tupeltype spreken wij ook van de *karakteriserende tabelconstraints* bij een tabeltype. Gegeven een tabeltype TT ligt dus vast: het kleinste karakteriserende tupeltype MT en een bijbehorende karakteriserende tabelconstraint TTC zodanig dat:

$$TT = \{D \mid D \in P(MT) \wedge TTC(D)\}.$$

Evenals bij tupeltypen zal over het algemeen niet de bedoeling zijn dat bij een tabeltype TT het bijbehorende kleinste karakteriserende tupeltype en een karakteriserende tabelconstraint worden 'gezocht'. Ook hier zal het uitgangspunt niet zijn een tabeltype, maar omgekeerd een (karakteriserend) tupeltype, waar bovenop dan tabelconstraints worden gedefinieerd. En hiermee ligt dan uiteraard een tabeltype vast.

Voorbeeld 3.11

Gegeven tupeltype MT van voorbeeld 3.9 en de karakteriserende tabelconstraint C2 van voorbeeld 3.10.
Het tabeltype TT5 wordt gedefinieerd door:

$$TT5 = \{D \mid D \in P(MT) \wedge C2(D)\}.$$

TT5 bevat 54 elementen: $\emptyset$, 7 elementen met precies één tupel, 18 elementen met precies 2 tupels, 20 elementen met precies 3 tupels, 8 elementen met precies 4 tupels.

Het is ook duidelijk dat voor TT4 van voorbeeld 3.9 geldt: TT4 $\subset$ TT5 en beide hebben hetzelfde kleinste karakteriserende tupeltype.

□

Voorbeeld 3.12

Gegeven tupeltype T-ART van voorbeeld 3.4 en de karakteriserende tabelconstraint C over P(T-ART), met $C = C1 \wedge C2 \wedge C3$ en

$$C1(D) = [\forall t1,t2 \in D[t_1(ano) = t_2(ano) \Rightarrow t_1 = t_2]].$$

$$C2(D) = [\forall t1,t2 \in D[t_1(gew) = t_2(gew) \Rightarrow t_1(pr) = t_2(pr)]].$$

$$C3(D) = [\#(\{t \mid t \in D \wedge t(kl) = rd\}) \leq \#(\{t \mid t \in D \wedge t(kl) = w\})].$$

Het tabeltype TT-ART wordt gedefinieerd door:

$$TT\text{-}ART = \{D \mid D \in P(T\text{-}ART) \wedge C(D)\}.$$

Dit tabeltype bevat zeer veel elementen. □

Het is theoretisch denkbaar dat een tabeltype TT gelijk is aan de machtsverzameling van het bijbehorende kleinste karakteriserende tupeltype MT, dus een tabeltype zonder enige tabelconstraints, dus zonder een karakteriserend tabelconstraint TTC (formeel: TTC = true). Als dan ook nog voor MT geldt dat dit een tupeltype is zonder enige tupelconstraints, dan is TT gelijk aan een filetype in Pascal. Een Pascal file-type is dus een wel heel speciaal geval van een tabeltype.

Wij beschikken nu over waardenverzamelingen van het tabeltype. Wij kunnen nu dan ook variabelen declareren van een tabeltype.

Voorbeeld 3.13

Met:

```
var A : T;
    B1,B2 : TT5;
    BA : TT-ART
endvar;
```

worden variabelen gedeclareerd van het tupeltype T (vb. 3.3) en het tabeltype TT5 (vb. 3.11) respectievelijk TT-ART (vb. 3.12).

Met het onderstaande programmadeel:

```
B1 := < > ;
A.a := 1; A.b := x; A.c := k;
B.1 := B1 un <A>;
A.a := 2; A.b := y; A.c := 1;
B.1 := B1 un <A>;
```

wordt bewerkstelligd dat de variabele B1 uiteindelijk twee tupels van het type T 'bevat'; dus dan is:

	a	b	c
B1 =	1	x	k
	2	y	1

In dit voorbeeld zijn de volgende syntax-elementen 'ad hoc' ingevoerd:

< >	:	geeft een verzameling weer
un	:	symbool voor de operatie 'vereniging van twee verzamelingen'.

3.3 ENKELE OPMERKINGEN OVER CONSTRAINTS

Bij een gegeven objectkarakterisering kunnen talloze tupel- en tabelconstraints worden bedacht. Definitie van een constraint heeft overigens alleen maar zin, als deze constraint wordt 'bewaakt'. Deze bewaking kan geschieden via de toepassingsprogramma's van de gebruikers, maar dit zal in het algemeen leiden tot een omslachtige en vooral onbetrouwbare werkwijze. Bewaking van constraints hoort dan ook te geschieden door het DBMS (database management systeem). Men moet echter niet onderschatten wat dit voor gevolgen heeft voor de omvang en complexiteit (en dus de duurte!) van een DBMS-pakket. Het is dan ook niet verwonderlijk dat de huidige beschikbare DBMS-pakketten op het gebied van constraints nauwelijks iets te bieden hebben.

Uit het bovenstaande moge wel duidelijk zijn, dat het in het algemeen raadzaam is bij de definitie van constraints een wijze beperking in acht te nemen.

In de voorafgaande paragrafen was sprake van zogeheten *statische constraints*. Dit zijn constraints, die beperkingen opleggen aan waarden, die 'op een bepaald moment' mogen optreden. In de loop van de tijd kunnen deze waarden echter veranderen.

Het spreekt vanzelf dat de nieuwe waarden dan ook weer aan de statische constraints moeten voldoen. Bij overgang op nieuwe waarden kunnen echter nog aparte 'overgangsbeperkingen' ofwel zogeheten *dynamische constraints* gelden. Een bekend voorbeeld hiervan is het volgende.

Het attribuut 'bs' (burgerlijke staat) van een persoon (uit het personenregister) kan de waarden ongehuwd (ongehuwd uiteraard te verstaan als nooit gehuwd geweest), gehuwd, gescheiden, weduwstaat hebben. Nu zullen niet alle overgangen tussen deze vier waarden zijn toegestaan, maar alleen die zoals aangegeven met 'ja' in onderstaande tabel.

bs oud / bs nieuw	ongehuwd	gehuwd	gescheiden	weduwstaat
ongehuwd	X	ja	nee	nee
gehuwd	nee	X	ja	ja
gescheiden	nee	ja	X	nee
weduwstaat	nee	ja	nee	X

Voor dynamische constraints geldt nog sterker dan voor statische constraints, dat de beschikbare DBMS-pakketten daar in het geheel niet op ingespeeld zijn.

In dit boek zullen verder alleen statische constraints worden besproken.

OPGAVEN

3.1 Geef de constraints C2 en C3 van voorbeeld 3.1 in woorden weer.

3.2 Welke van onderstaande constraints is equivalent met C2 van voorbeeld 3.1?

$Ca(t) = [t(ano) < 75 \wedge t(gew) > 20]$

$Cb(t) = [t(ano) < 75 \vee t(gew) > 20]$

$Cc(t) = [\neg(t(ano) < 75) \vee t(gew) > 20]$

$Cd(t) = [t(gew) \leq 20 \Rightarrow t(ano) \geq 75]$

3.3 Ga na dat door C1 in voorbeeld 3.1 inderdaad 10 tupels van $\Pi(F)$ worden 'uitgeschakeld'.

3.4 Geef de volgende constraints over Π(ART) (ART van voorbeeld 2.6c) formeel weer.
   a. Als de prijs > 400 is, mag de voorraad niet groter dan 1000 zijn.
   b. Bij een prijs < 100 èn een gewicht > 100 mag de kleur niet rood zijn.
   c. Bij een prijs < 100 mag het gewicht niet > 100 en de kleur niet rood zijn.
   d. Bij een prijs < 100 moet het gewicht ≤ 100 of de kleur niet rood zijn.

3.5 Gegeven onderstaande objectkarakterisering OPN (opname van een patiënt in een ziekenhuis.

	Toelichting
OPN = {(pnr,{1...100000}),	patiëntnummer
(indat,{700101...991231}),	datum opname
(opnr,charstring30),	opnamereden
(snr,{1...100}),	nummer opnamespecialist
(snm,charstring25),	naam "

(pnm,charstring25),	naam patiënt
(pwpl,charstring25),	woonplaats patiënt
(padr,charstring30),	adres patiënt
(uitdat,{700101..991231}),	datum vertrek
(gbd,{8800101...9991231}),	geboortedatum patiënt
(anr,{1...100})}	nummer afdeling

Bedenk enkele tupelconstraints over Π(OPN) en geef deze formeel en in woorden weer.

3.6 Ga na dat in voorbeeld 3.2 inderdaad geldt:

$Y = \{t \mid t \in \Pi(F') \wedge TC2(t)\}$.

3.7 Gegeven het tupeltype T, bepaald door de onderstaande objectkarakterisering F en de constraints C1, C2 en C3 over Π(F).

$F = \{(a,\{1,2,3,4\}),(b,\{5,6,7\}),(c,\{1,k,x,f\})\}$

$C1(t) = [t(a) = 3 \Rightarrow (t(b) = 6 \wedge t(c) = 1]$

$C2(t) = [t(c) \neq f \Rightarrow (t(b) \leq 5)]$

$C3(t) = [t(a) + t(b) \leq 9]$

a. Bepaal de kleinste objectkarakterisering KF van T.
b. Geef een enumeratieve beschrijving van T.
c. Is de conjunctie van C1 ∧ C2 ∧ C3 een echte tupelconstraint? Zo nee, bepaal dan welke attribuutconstraint(s) in deze conjunctie bevat zijn. Welk verband bestaat er tussen de antwoorden op vragen a en c?

3.8 Ga na welk van onderstaande verzamelingen Vi tupeltypen zijn. Zo ja, bepaal dan de kleinste objectkarakterisering Ki en verder |Vi| en |Π(Vi)|.

a. $V1 = \{\{(a,5),(b,5)(c,5)\},\{(a,5),(b,3),(c,2)\}\}$

b. $V2 = \{(a,5),(b,5),(c,5)\}$

c. $V3 = \{\{(a,5),(b,5),(c,5)\},\{(a,\{5,4\}),(b,3),(c,2)\}\}$

d. $V4 = \{\{(a,5),(c,5)\},\{(a,5),(b,3),(c,2)\}\}$

e. $V5 = \{f \mid f \text{ is een functie} \wedge dom(f) \in \{a,b,c\} \wedge rng(f) \subset \{2,3,5\}\}$

f. $V6 = \{f \mid f \text{ is een functie} \wedge dom(f) = \{a,b,c\} \wedge rng(f) \subset \{2,3,5\}\}$

g. $V7 = \{f \mid f \text{ is een functie} \wedge dom(f) = \{a,b,c\} \wedge rng(f) = \{2,3,5\}\}$

h. $V8 = \{f \mid f \text{ is een functie} \wedge dom(f) \subset \{a,b,c\} \wedge rng(f) = \{2,3,5\}\}$

i. $V9 = \{f \mid f \text{ is een functie} \wedge dom(f) = \{a,b,c\} \wedge rng(f) \in \{\{2,3\},\{2\},\{2,3,5\}\}$

3.9 t is een variabele van het tupeltype T van voorbeeld 3.3.

a. Ga na, welke van onderstaande beweringen strijdig is met de gegeven definitie van T.
   - a1. t(a) = 4 * t(c)
   - a2. t(a) = 4 * t(c) ∧ t(b) = z
   - a3. t(a) ≠ 4 * t(c)
   - a4. t(b) = y ∧ t(c) = x.

b. Welke van onderstaande programmadelen zijn toegestaan volgens de definitie van T.
   - b1. t(a) := 1; t(b) := z; t(c) := x;
     t(b) := x; t(a) := t(a) + 1
   - b2. t(b) := y; t(c) := 1; t(a) := 3 * t(c);
     t(a) := t(a) + 1.

3.10 Tabeltype X is als volgt gedefinieerd:

$X = \{\emptyset, D1, D2\}$ (D1,D2 van voorbeeld 3.6).

Bepaal het kleinste karakteriserende tupeltype MT van X en ook nog een ander karakteriserend tupeltype van X.

3.11 Deze opgave heeft betrekking op voorbeeld 3.7.

a. Bepaal |TT1|, |TT2| en |TT3|.

b. Bepaal het kleinste karakteriserend tupeltype van TT1 en van TT3.

c. Welke van onderstaande verzamelingen A-G zijn tabeltypen en welke van deze tabeltypen zijn gelijk aan TT1, of TT2 of TT3?

$A = \emptyset \cup \{\Pi(F), Y, T\}$; $B = \{\emptyset\} \cup \{\Pi(F), Y, T\}$; $C = \{\emptyset, Y, T\}$;

$D = \{\emptyset, \{Y\}, T\}$; $E = TT1 \setminus Y$; $H = TT3 \setminus TT1$; $G = H \cup TT2$.

3.12 Bewijs (enkele regels onder voorbeeld 3.8): $KT \supset MT$.

3.13 Geef de constraints van voorbeeld 3.10 verbaal weer.

3.14 Welke van onderstaande tabelconstraints is equivalent met C2 van voorbeeld 3.10?

$Ca(D) = [\forall t1,t2 \in D[t1(a) \neq t2(a)]$

$Cb(D) = [\forall t1,t2 \in D[t1(a) \neq t2(a) \vee t1 = t2]$

$Cc(D) = [\forall t1,t2 \in D[t1 \neq t2 \Rightarrow t1(a) \neq t2(a)]$.

3.15 Geef de volgende tabelconstraints over P(T-ART) formeel weer (T-ART van voorbeeld 3.4):
a. Het aantal tupels met een gewicht > 20 èn een prijs > 300 mag niet groter zijn dan 10.
b. De totale voorraad van rode tafels mag niet minder dan 10 en niet groter dan 100 zijn.
c. Van rode tafels mogen hoogstens 10 verschillende prijzen voorkomen.
d. Gelijk gewicht èn gelijke prijs houdt ook in gelijke kleur.

3.16 Ga na dat TT5 van voorbeeld 3.11 inderdaad 54 elementen bevat.

3.17 Het tupeltype T-OPN is gedefinieerd door:

$T\text{-}OPN = \{t \mid t \in \Pi(OPN) \wedge C(t)\}$, waarbij OPN uit opgave 3.5

èn $C(t) := [t(uitdat) > t(indat) \wedge t(opnr) = informaritis \Rightarrow$

$t(snm) = brakes]$.

Bedenk tabelconstraints over P(T-OPN) en geef deze formeel èn verbaal weer.

3.18 TA is een variabele van het tabeltype TT-ART voor voorbeeld 3.12.
a. Ga na, welke van onderstaande beweringen strijdig is met de gegeven definitie van TT-ART.
a1. TA = V, waarbij (in tabelvorm weergegeven)

V =

ano	anm	pr	gew	vrd	kl
8	bank	20	25	0	rd
92	kast	150	150	800	rd
63	stoel	12	10	10	rd
103	bank	60	80	15	w
5	kast	125	15	20	w

a2. $TA \supset V$ (V als sub a1)
a3. $TA \subset V$ (V als sub a1)
a4. $|TA| = 400$
a5. $|TA| = 0$.

b. Welke delen van onderstaand programma zijn toegestaan met de gegeven definitie van TT-ART.

```
TA := ∅;
TA := TA un{{(ano,90),(anm,lamp),(pr,30),(gew,7),
            (vrd,82),(kl,w)}}
TA := TA un{{(ano,60),(anm,lamp),(pr,30),(gew,7),
            (vrd,82),(kl,rd)}}
TA := TA un{{(ano,40),(anm,stoel),(pr,40),(gew,7),
            (vrd,120),(kl,rd)}}.
```

4 AFHANKELIJKHEDEN; SLEUTEL

4.1 AFHANKELIJKHEDEN

In dit hoofdstuk zullen wij ingaan op een belangrijke klasse van tabelconstraints, en wel de klasse van afhankelijkheden.
Voor dit gehele hoofdstuk zullen wij aannemen dat:

TT	een tabeltype is,
A	de verzameling van alle attributen van TT is,
A_1, A_2, A_3	deelverzamelingen van A zijn.

Ruwweg kunnen wij zeggen dat de attributen(deel)verzameling A_2 afhankelijk is van de attributen(deel)verzameling A_1, als de waarden van de attributen van A_2 'vastliggen' door de waarden van de attributen van A_1. Dit 'vastliggen' gaan wij nu definiëren.

Definitie 4.1
(niet-formeel) *A_2 is afhankelijk van A_1 onder TT* als voor elke tabel T van TT geldt: als een tweetal tupels van T gelijke waarden heeft voor A_1, dan zal dit tweetal ook gelijke waarden hebben voor A_2.
(formeel) *A_2 is afhankelijk van A_1 onder TT* $\overset{D}{\equiv}$

$$\forall T \in TT\ [\ \forall t1, t2 \in T\ [\ t_1 \lceil A_1 = t_2 \lceil A_1 \Rightarrow t_1 \lceil A_2 = t_2 \lceil A_2\ ]\ ].$$ □

Notatie: $A_1 \underset{TT}{\rightarrow} A_2$.

Als duidelijk is (uit de context) onder welk tabeltype een afhankelijkheid geldt, dan wordt de aanduiding van het desbetreffende tabeltype meestal weggelaten, en dus ook in de notatie $A_1 \rightarrow A_2$.

Uit de definitie volgt onmiddellijk:
Als A_2 afhankelijk is van A_1, dan geldt voor elke tabel T van TT, dat de paren $(t\lceil A_1, t\lceil A_2)$ van alle tupels t van T een functie vormen èn omgekeerd.

Meer formeel:

$$A_1 \rightarrow A_2 \Leftrightarrow \forall T \in TT\ [\ \{(t \lceil A_1, t \lceil A_2) \mid t \in T\}\ \text{is een functie}\ ].$$

Hiermee moge duidelijk zijn, waarom in de database-literatuur dikwijls het begrip afhankelijkheid wordt aangegeven met 'functionele afhankelijkheid' (Engels: functional dependency).

Voorbeeld 4.1

Voor dit voorbeeld roepen wij in herinnering (de constructie van) tabeltype TT-ART (voorbeelden 2.6c, 3.1, 3.4, 3.12).

- objectkarakterisering ART
  ART = {(ano,{1...100}),(anm,charstring 20),(pr,{10...500}),
  (gew,{1...300}),(vrd,{0...10000}),(kl,{rd,w,bl})}.
- tupelconstraints C2 en C3 over Π(ART)
  $C2(t) := [t(ano) < 75 \Rightarrow t(gew) > 20]$
  $C3(t) := [t(pr) \times t(vrd) < 100000]$
- tupeltype T-ART
  $T\text{-}ART = \{t \mid t \in \Pi(ART) \wedge (C2 \wedge C3)(t)\}.$
- tabelconstraints C1, C2 en C3 over P(T-ART)
  $C1(D) := [\forall t1,t2 \in D\, [t_1(ano) = t_2(ano) \Rightarrow t_1 = t_2]]$
  $C2(D) := [\forall t1,t2 \in D\, [t_1(gew) = t_2(gew) \Rightarrow t_1(pr) = t_2(pr)]]$
  $C3(D) := [\#(\{t \mid t \in D \wedge t(kl) = rd\}) \leq \#(\{t \mid t \in D \wedge t)kl) = w\})].$
- tabeltype TT-ART
  $TT\text{-}ART = \{D \mid D \in P(T\text{-}ART) \wedge (C1 \wedge C2 \wedge C3)(D)\}.$

Onder TT-ART is blijkbaar {pr} afhankelijk van {gew}. Immers tabelconstraint C2 is geldig voor elk element van TT-ART, en deze tabelconstraint C2 is equivalent met:

$$\forall t1,t2 \in D\, [(t_1 \lceil \{gew\} = t_2 \lceil \{gew\} \Rightarrow t_1 \lceil \{pr\} = t_2 \lceil \{pr\}].$$

Verder volgt ook nog uit de tabelconstraint C1 de afhankelijkheid {ano} $\rightarrow$ ATT-ART, waarbij ATT-ART de verzameling is van alle attributen van TT-ART. □

Als $A_1 \rightarrow A_2$, dan noemen wij het geordende paar (A_1,A_2) een *afhankelijkheid* (onder TT). De eerste coördinaat van een afhankelijkheid wordt ook wel *het linkerlid* ofwel *de determinant* van de afhankelijkheid genoemd. De tweede coördinaat heet ook wel *het rechterlid* ofwel *de gedetermineerde* van de afhankelijkheid.

In de literatuur komt men dikwijls zinsneden tegen zoals 'time-varying dependencies'. Dit time-varying aspect van een afhankelijk-

heid heeft nu formeel gestalte gekregen in dat deel van definitie 4.1 waar sprake is van *elke* tabel van TT of meer formeel $\forall T \in TT$.

Belangrijk is het onderscheid tussen zogeheten triviale en niet-triviale afhankelijkheden.

Definitie 4.2
(niet-formeel) A_2 is *triviaal afhankelijk* van A_1 als A_2 een deelverzameling is van A_1.
(formeel) A_2 is *triviaal afhankelijk* van $A_1 \overset{D}{\equiv} A_2 \subset A_1$. □

Het is eenvoudig in te zien dat uit $A_2 \subset A_1$ volgt $A_1 \rightarrow A_2$. De naam triviale *afhankelijkheid* is dus terecht gekozen. Het triviale is gelegen in het feit, dat deze afhankelijkheid geldt, ongeacht de waarden, die de attributen kunnen aannemen. Een verzameling van attributen is dus altijd triviaal afhankelijk van elke 'omvattende' attributenverzameling.

'Afhankelijk van' is een *transitieve* relatie (tussen de deelverzamelingen van de attributenverzameling A), zoals blijkt uit:

Stelling 4.1 (Transitiviteitsstelling)
Als $A_1, A_2, A_3 \subset A$ èn $A_1 \rightarrow A_2$ èn $A_2 \rightarrow A_3$, dan geldt: $A_1 \rightarrow A_3$.

Bewijs: Uit $A_1 \rightarrow A_2$ volgt:

(1) $\forall T \in TT\,[\forall t1,t2 \in T\,[t_1\lceil A_1 = t_2\lceil A_1 \Rightarrow t_1\lceil A_2 = t_2\lceil A_2]]$.

Uit $A_2 \rightarrow A_3$ volgt:

(2) $\forall T \in TT\,[\forall t1,t2 \in T\,[t_1\lceil A_2 = t_2\lceil A_2 \Rightarrow t_1\lceil A_3 = t_2\lceil A_3]]$.

Uit (1) en (2) volgt:

(3) $\forall T \in TT\,[\forall t1,t2 \in T\,[t_1\lceil A_1 = t_2\lceil A_1 \Rightarrow t_1\lceil A_3 = t_2\lceil A_3]]$.

Dus: $A_1 \rightarrow A_3$. □

'Afhankelijk van' is ook een *reflexieve* relatie (tussen de deelverzamelingen van de attributenverzameling A. Er geldt namelijk voor elke $A_1 \subset A$: $A_1 \rightarrow A_1$ (triviale afhankelijkheid).

'Afhankelijk van' is *geen* equivalente relatie (tussen de deelverzamelingen van de attributenverzameling A). Met andere woorden: uit $A_1 \rightarrow A_2$ volgt niet $A_2 \rightarrow A_1$.
Geldt echter voor A_2 en A_1 zowel $A_1 \rightarrow A_2$ als $A_2 \rightarrow A_1$, dan zeggen wij dat A_1 *equivalent is met* A_2. Notatie: $A_1 \leftrightarrow A_2$.

Nu volgen enkele stellingen voor het afleiden van afhankelijkheden uit gegeven afhankelijkheden. Overigens was stelling 4.1 al zo'n stelling.

Stelling 4.2 (Rechter verenigingsstelling)

Als $A_1 \to A_2$ èn $A_1 \to A_3$ dan geldt $A_1 \to A_2 \cup A_3$.

Bewijs: Uit $A_1 \to A_2$ volgt:

(1) $\forall T \in TT\, [\, \forall t1,t2 \in T\, [\, t_1 \lceil A_1 = t_2 \lceil A_1 \Rightarrow t_1 \lceil A_2 = t_2 \lceil A_2 \,]\,]$.

Uit $A_1 \to A_3$ volgt:

(2) $\forall T \in TT\, [\, \forall t1,t2 \in T\, [\, t_1 \lceil A_1 = t_2 \lceil A_1 \Rightarrow t_1 \lceil A_3 = t_2 \lceil A_3 \,]\,]$.

Uit (1) en (2) volgt:

(3) $\forall T \in TT\, [\, \forall t1,t2 \in T\, [\, t_1 \lceil A_1 = t_2 \lceil A_1 \Rightarrow t_1 \lceil A_2 \cup A_3 = t_2 \lceil A_2 \cup A_3 \,]\,]$.

Dus: $A_1 \to A_2 \cup A_3$. □

Stelling 4.3 (Linker uitbreidingsstelling)

Als $A_1 \to A_2$ èn $A_3 \supset A_1$ dan geldt $A_3 \to A_2$.

Bewijs: Uit $A_3 \supset A_1$ volgt (triviale afhankelijkheid)

(1) $A_3 \to A_1$.

Uit (1) en $A_1 \to A_2$ volgt (met stelling 4.1): $A_3 \to A_2$. □

Stelling 4.4 (Rechter inkrimpingsstelling)

Als $A_1 \to A_2$ èn $A_3 \subset A_2$ dan geldt $A_1 \to A_3$.

Bewijs: Wordt overgelaten aan de lezer (zie opgave 4.1). □

Stelling 4.5 (Verenigingsstelling)

Als $A_1 \to A_2$ èn $A_3 \to A_4$ dan geldt $A_1 \cup A_3 \to A_2 \cup A_4$.

Bewijs: Uit $A_1 \to A_2$ volgt (met stelling 4.3):

(1) $A_1 \cup A_3 \to A_2$.

Uit $A_3 \to A_4$ evenzo:

(2) $A_1 \cup A_3 \to A_4$.

Uit (1) en (2) met stelling 4.2: $A_1 \cup A_3 \to A_2 \cup A_4$. □

Als A_2 afhankelijk is van A_1, dan hoeft dat nog niet te betekenen, dat er geen echte deelverzameling A_{11} van A_1 is zó dat $A_{11} \rightarrow A_2$. Is er echter niet zo'n deelverzameling, dan zeggen wij dat A_2 van de volledige A_1 afhankelijk is. Een en ander wordt nu vastgelegd in:

Definitie 4.3
(niet-formeel) A_2 is *afhankelijk van de volledige* A_1 als A_2 afhankelijk is van A_1 en A_2 niet afhankelijk is van een echte deelverzameling van A_1.
(formeel) A_2 is *afhankelijk van de volledige* $A_1 \overset{D}{\equiv}$

$\overset{D}{\equiv} A_1 \rightarrow A_2$ èn $\forall D \subset A_1 [D \neq A_1 \Rightarrow \rceil(D \rightarrow A_2)]$. □

Notatie: $A_1 \overset{V}{\rightarrow} A_2$.

Een linkeruitbreidingsstelling kan uiteraard niet gelden voor volledige afhankelijkheid, maar ook is er geen rechter inkrimpingsstelling voor volledige afhankelijkheid.

Dus uit $A_1 \overset{V}{\rightarrow} A_2$ volgt niet: $\forall a \in A_2 [A_1 \overset{V}{\rightarrow} \{a\}]$.

Het kan zelfs voorkomen dat, niettegenstaande $A_1 \overset{V}{\rightarrow} A_2$ geldt, voor geen enkele $a \in A_2$ geldt $A_1 \overset{V}{\rightarrow} \{a\}$. Een tegenvoorbeeld is het volgende:

Voorbeeld 4.2

Tabeltype TT6 bevat uitsluitend de hieronder weergegeven elementen T1 en T2. {a1,a2,a3,a4,a5} is de attributenverzameling ATT6 van TT6.

T1 =

a1	a2	a3	a4	a5
1	4	7	p	s
1	6	7	p	s
1	4	9	p	s
2	5	9	q	t
3	5	9	p	t
3	4	7	p	s

T2 =

a1	a2	a3	a4	a5
2	6	7	p	t
2	5	7	p	s
1	4	7	q	t
1	4	9	q	t
2	5	9	p	s

De lezer ga na dat geldt:

$\{a1,a2\} \overset{V}{\rightarrow} \{a4,a5\}$
$\{a1\} \rightarrow \{a4\}$
$\{a2\} \rightarrow \{a5\}$ □

Ten aanzien van volledige afhankelijkheid geldt wel de volgende stelling:

Stelling 4.6

Als (1) $A_1,A_2,A_{21} \subset A$ èn (2) $A_1 \rightarrow A_2$ èn (3) $A_{21} \subset A_2$ èn (4) $A_1 \overset{V}{\rightarrow} A_{21}$ dan geldt: $A_1 \overset{V}{\rightarrow} A_2$.

Bewijs. Stel (5) $D \subset A_1$ èn (6) $D \rightarrow A_2$.

Uit (6) en (3) volgt:

(7) $D \rightarrow A_{21}$.

Uit (7), (4) en (5) volgt:

(8) $D = A_1$.

Dus ($D \subset A_1$ èn $D \rightarrow A_2$) $\Rightarrow D = A_1$. Dus $A_1 \overset{V}{\rightarrow} A_2$. □

Een direct gevolg van stelling 4.6 is dat als de volledige afhankelijkheid (A_1,A_{21}) wordt 'uitgebreid' tot de afhankelijkheid (A_1,A_2), waarbij $A_{21} \subset A_2$, dan is de afhankelijkheid (A_1,A_2) ook weer volledig. Dus bij 'uitbreiding' van het rechterlid blijft volledige afhankelijkheid behouden.

4.2 SLEUTEL

Het bijzondere van de afhankelijkheid ({ano},A) in voorbeeld 4.1 is, dat de verzameling van alle attributen afhankelijk is van {ano}.

Een attributenverzameling, waarvan de gehele attributenverzameling afhankelijk is, noemen wij een *uniek identificerende* attributenverzameling van TT. Als {ano} uniek identificerend is, dan zullen ook alle $D \supset$ {ano} uniek identificerend zijn.

We zullen nu het begrip sleutel invoeren als een bijzondere, uniek identificerende attributenverzameling.

Definitie 4.4

(niet-formeel) A_1 is *sleutel* van TT als de gehele attributenverzameling A afhankelijk is van de volledige A_1.

(formeel)

$$A_1 \text{ is } \textit{sleutel} \text{ van TT} \overset{D}{\equiv} A_1 \xrightarrow[TT]{V} A. \qquad □$$

Opmerking

In de literatuur bestaat geen eenduidige opvatting over het begrip

sleutel (key). Velen definiëren sleutel (op min of meer duidelijke manier) als in definitie 4.4. Er zijn echter ook auteurs die sleutel definiëren in de zin van alleen uniek identificerend.

Uiteraard kan een tabeltype meer dan één sleutel hebben. We spreken daarom van de *sleutelverzameling* van een tabeltype. Hierbij is het zinloos onderscheid te maken tussen zogeheten primaire en secundaire sleutels, zoals door diverse auteurs wordt gedaan.

Voorbeeld 4.3

Gegeven is tabeltype TT-ORDER met K-ORDER als kleinste objectkarakterisering en de afhankelijkheden f1 tot en met f7.

K-ORDER =	Toelichting
{(onr,{1...100000}),	ordernummer
(dat,{700101...991231}),	datum
(lnr,{1...100}),	leveranciersnummer
(arc,{1...100}),	artikelcode
(hoev,{1...100000}),	hoeveelheid
(lnm,charstring 25),	leveranciersnaam
(anm,charstring 20),	artikelnaam
(ladr,charstring 30),	leveranciersadres
(lwpl,charstring 20),	leverancierswoonplaats
(bestadr,charstring 30),	besteladres
(pr,{10...5000}}	prijs per stuk.

f1 : $\{onr\} \rightarrow A$ (A = dom(K-ORDER))

f2 : $\{lnr,arc,dat\} \overset{V}{\rightarrow} A$

f3 : $\{lnr\} \leftrightarrow \{lnm\}$

f4 : $\{arc\} \leftrightarrow \{anm\}$

f5 : $\{lnr\} \rightarrow \{ladr,lwpl\}$

f6 : $\{lnr,arc\} \overset{V}{\rightarrow} \{pr\}$

f7 : $\{arc\} \rightarrow \{bestadr\}$.

Uit f1 en f2 volgt dat {onr} en {lnr,arc,dat} sleutels zijn. Uit f1 t/m f4 volgt dat de sleutelverzameling s is:
s = {{onr},{lnr,arc,dat},{lnm,arc,dat},{lnr,anm,dat},{lnm,anm,dat}}.

□

Het is eenvoudig in te zien, dat sleutels van eenzelfde tabeltype onderling equivalent zijn. Omdat elke sleutel 'minimaal' is, bevat een sleutel geen echte deelverzameling die equivalent is met deze sleutel. Wel kan het voorkomen dat echte deelverzamelingen van twee ver-

schillende sleutels equivalent zijn, zoals {lnr} en {lnm} in voorbeeld 4.3. Ook deze equivalentie zal echter niet meer kunnen voorkomen, als sprake is van zogeheten genormaliseerde tabeltypen (onderwerp van het volgende hoofdstuk).

OPGAVEN

4.1 Geef het bewijs van stelling 4.4

4.2 Bewijs: Als $A_1, A_2 \subset A$, dan geldt
$(A_1 \rightarrow A_2) \Leftrightarrow (\forall a \in A_2 [A_1 \rightarrow \{a\}]$.

4.3 (opgave bij voorbeeld 4.2)
a. Ga na dat de genoemde afhankelijkheden inderdaad gelden.
b. Bepaal de sleutelverzameling van TT6.
c. Los X op uit:
c1. $X \rightarrow \{a1\}$; c2. $X \rightarrow \{a3\}$; c3. $\{a1,a2\} \overset{V}{\rightarrow} X$;
c4. $\{a2,a3\} \overset{V}{\rightarrow} X$.
d. Bepaal alle deelverzamelingen van ATT6 die equivalent zijn met {a1,a2}.

4.4 Bewijs dat $A_1 \overset{V}{\rightarrow} A_2$ equivalent is met $\forall D \in A_1 [(D \rightarrow A_2) \Leftrightarrow (D = A_1)]$.

4.5 (opgave bij voorbeeld 4.3)
a. Welke van onderstaande beweringen is waar?
a1. {onr} → {arc,hoev}; a2. {onr,lnr} → {arc,hoev};
a3. {lnr,arc} → {lnm,anm}; a4. {lnr,arc} → {lnm,amn,pr};
a5. {onr,lnr} $\overset{V}{\rightarrow}$ {ladr,lwpl}; a7. {anm,lnr} $\overset{V}{\rightarrow}$ {hoev}.
b. Geef commentaar op de volgende beweringen.
b1. Per order kunnen meerdere leveranciers optreden.
b2. Elk artikel heeft een vaste prijs.
b3. Per order wordt het besteladres vastgesteld.

5 NORMALISATIE

5.1 GENORMALISEERD TABELTYPE

Tot nu toe hebben wij dikwijls gesproken over attributen van een object (karakterisering), maar eigenlijk niet 'bekeken' of elk van de genoemde attributen wel echt bij dat object 'thuishoort'. Als wij bijvoorbeeld de attributen van de objectkarakterisering K-ORDER van voorbeeld 4.3 onder de loep nemen, dan ligt het voor de hand dat attributen als 'onr', 'dat', 'hoev' er echt in thuishoren, terwijl dit niet zo duidelijk is voor attributen als 'lnm', 'ladr', 'lwpl', 'anm'. Of een bepaald attribuut in een objectkarakterisering thuishoort, moet uiteraard geen nietes-welles spelletje worden. Wij moeten formeel-duidelijke criteria aanleggen, waarmee te toetsen valt of een attribuut wel of niet tot een karakterisering behoort. Deze criteria dienen te zelfder tijd semantisch duidelijk te zijn, dat wil zeggen: ze dienen te appeleren aan een intuïtieve 'opdeling' van de attributen over de objectkarakteriseringen.

Met het begrip afhankelijkheid en meer in het bijzonder het begrip sleutel (in het vorige hoofdstuk) hebben wij een goede basis gecreeerd voor bedoelde criteria. Met behulp van deze begrippen zullen wij nu het begrip 'genormaliseerd tabeltype' vastleggen.

Definitie 5.1
Stel A_1 en A_2 zijn deelverzamelingen van de attributenverzameling van het tabeltype TT èn SL is de sleutelverzameling van TT.
(niet-formeel) TT is een *genormaliseerd tabeltype* als voor elke niet-triviale afhankelijkheid $A_1 \rightarrow A_2$ geldt dat A_1 een sleutel bevat.
(formeel)
TT is een *genormaliseerd tabeltype* $\stackrel{D}{\equiv}$ voor elke niet-triviale afhankelijkheid $A_1 \underset{TT}{\rightarrow} A_2$ geldt: $[\exists s \in SL[s \subset A_1]]$. □

Voorbeeld 5.1

Tabeltype TT-ORDER van voorbeeld 4.3 is niet genormaliseerd. Er geldt namelijk {lnr } → {lnm } en dit is een niet-triviale afhankelijkheid, waarvan het linkerlid geen sleutel bevat. Overigens is f3 niet

de enige afhankelijkheid, waarvoor het bovenstaande geldt.

Om, met behoud van 'informatiewaarde' tot een genormaliseerde opzet te komen, zal het nodig zijn om met meer dan één tabeltype te werken. In dit geval kunnen daarvoor de vier onderstaande tabeltypen dienen:

1. Tabeltype TT-ORDER-N met K-ORDER-N als kleinste objectkarakterisering en de afhankelijkheden f1 en f2:

   K-ORDER-N =
   {(onr,{1...100000}),
   (dat,{700101...991231}),
   (lnr,{1...100}),
   (arc,{1...100}),
   (hoev,{1...100000})}.

   f1: {onr} $\rightarrow$ AN

   f2: {lnr,arc,dat} $\overset{V}{\rightarrow}$ AN

   waarbij AN de attributenverzameling van TT-ORDER-N is.
   Uit het voorgaande volgt dat

   S-ORDER-N = {{onr},{lnr,arc,dat}},

   waarbij S-ORDER-N de sleutelverzameling van TT-ORDER-N is.

2. Tabeltype TT-LEV-N met K-LEV-N als kleinste objectkarakterisering en de afhankelijkheden f3 en f5:

   K-LEV-N =
   {(lnr,{1...100}),
   (lnm,charstring 25),
   (ladr,charstring 30),
   (lwpl,charstring 20)}.

   f3: {lnr} $\leftrightarrow$ {lnm}

   f5: {lnr} $\rightarrow$ {ladr,lwpl}.

   Uit het voorgaande volgt dat S-LEV-N = {{lnr},{lnm}}, waarbij S-LEV-N de sleutelverzameling van TT-LEV-N is.

3. Tabeltype TT-ART-N met K-ART-N als kleinste objectkarakterisering en de afhankelijkheden f4 en f7:

   K-ART-N =
   {(arc,{1...100}),
   (anm,charstring 20),
   (bestadr,charstring 30)}.

   f4: {arc} $\leftrightarrow$ {anm}

   f7: {arc} $\rightarrow$ {bestadr}.

Uit het voorgaande volgt dat S-ART-N = {{arc},{anm}}, waarbij S-ART-N de sleutelverzameling van TT-ART-N is.

4. Tabeltype TT-ASS-N (ASSortiment) met K-ASS-N als kleinste objectkarakterisering en de afhankelijkheid f6:

   K-ASS-N =
   {(lnr,{1...100}),
   (arc,{1...100}),
   (pr,{10...5000})}.

   f6: {lnr,arc} → {pr}.

   Uit het voorgaande volgt dat S-ASS-N = {{lnr,arc}}, waarbij S-ASS-N de sleutelverzameling is van TT-ASS-N. □

In het voorgaande voorbeeld is sprake van genormaliseerde opzet, *met behoud van informatiewaarde*. Voor dit behoud van informatiewaarde is in ieder geval nodig dat de attributen en de afhankelijkheden van het ongenormaliseerde tabeltype, waarvan we uitgaan (zoals in het voorbeeld TT-ORDER), allemaal moeten zijn 'terug te vinden' in de genormaliseerde opzet. Met name geldt dit voor de attributen en afhankelijkheden, die uit het ongenormaliseerde tabeltype moeten 'verdwijnen', om dit tabeltype zelf te 'normaliseren'. (In het voorbeeld zijn de 'te verdwijnen' attributen: lnm, ladr, lwpl, anm, pr, bestadr. Met deze attributen verdwijnen dan ook de afhankelijkheden f3 tot en met f7.) Deze attributen zullen dan een plaats moeten vinden in bestaande tabeltypen of in nieuw te definiëren tabeltypen. Om te bereiken dat genoemde attributen en afhankelijkheden zodanig worden ondergebracht dat alleen genormaliseerde tabeltypen ontstaan, zal het nodig zijn dat elke afhankelijkheid f zodanig wordt 'ondergebracht' in een tabeltype TT dat het linkerlid van f een sleutel bevat van TT.

Behoud van informatiewaarde betekent ook dat bij elke tabel T-ORDER-N van TT-ORDER-N een tabel van respectievelijk TT-LEV-N, TT-ART-N, TT-ASS-N beschikbaar dient te zijn, met de bij elk ordertupel van T-ORDER-N behorende informatie over respectievelijk leverancier, artikel, assortiment.

Dit verband tussen tupels van tabellen van verschillende tabeltypen dient nog formeel vastgelegd te worden. Daartoe kunnen wij pas overgaan na invoering (in het volgende hoofdstuk) van het zogeheten databasetype. Voorlopig spreken wij af, dat genoemde verbanden vastliggen door gelijke attribuutnamen: 'lnr' in K-ORDER-N en in K-LEV-N, 'arc' in K-ORDER-N en in K-ART-N, 'lnr' en 'arc' in K-ORDER-N en in K-ASS-N.

Toen in 1970 door Codd het normalisatiebegrip werd ingevoerd, maakte hij onderscheid tussen eerste, tweede en derde normaalvorm. 1NF, 2NF, 3NF zijn hiervoor de gebruikelijke afkortingen (NF =

Normal Form). Daarna zijn er nog (door Fagin en anderen) 'hogere' normaalvormen ontwikkeld, zoals 4NF en 5NF.

Als *argument* voor normalisatie werd en wordt nog steeds aangevoerd, dat daarmee redundantie in opslag van gegevens zou worden vermeden.

Op de (verschillende) normaalvormen zullen wij in paragraaf 5.2 ten dele ingaan en de argumentatie voor normalisatie zal worden besproken in paragraaf 5.3.

5.2 NORMAALVORMEN

We zullen in feite alleen de eerste normaalvorm bespreken en ons voor de overige normaalvormen tot enkele opmerkingen beperken.

De eerste normaalvorm is de meest merkwaardige en meest onduidelijk gedefinieerde normaalvorm, die overigens tot op heden nog steeds als zodanig wordt gebruikt door de meeste auteurs. Een tabeltype heet dan in eerste normaalvorm, als alle attribuutwaarden 'atomair' zijn. In plaats van 'atomair' wordt ook gesproken van 'niet verder opsplitsbaar'. Men drukt dit ook wel uit door te zeggen, dat er geen zogeheten 'repeating groups' in een object-occurrence mogen voorkomen, zoals bijvoorbeeld meerdere waarden voor het attribuut 'anm' in een tupel van het tupeltype T-ART van voorbeeld 3.4. In onze formele opzet is deze 'meerwaardigheid' van een attribuut in een tupel per definitie uitgesloten. Door de gekozen definitie van tupeltype wordt in een tupel aan elk attribuut precies één waarde toegevoegd uit de verzameling van voor dat attribuut mogelijke waarden. Dit wil niet zeggen, dat in deze verzameling van attribuutwaarden geen waarden zouden kunnen voorkomen, die de 'schijn' hebben van 'meerwaardigheid'. Een voorbeeld moge dit laatste verduidelijken.

Voor het attribuut 'anm' (artikelnaam) zou de volgende waardenverzameling W1 = {lepel,vork,mes} denkbaar zijn, maar ook W2 = {lepel,vork,mes,lepel-vork, vork-mes, lepel-vork-mes}. W2 bestaat niet uit 3, maar uit 6 elementen, en is dus niet gelijk aan W1. Zowel W1 als W2 kunnen worden gekozen als waardenverzameling van attribuut 'anm'. Als echter W1 gekozen is, kunnen geen waarden als 'lepel-vork' of 'vork-mes' of 'lepel-vork-mes' voor het attribuut anm in een tupel optreden. Of een bepaalde waarde wel of niet tot een waardenverzameling behoort, is geheel ter beoordeling van de gebruiker, dus van degene die verantwoordelijk is voor de betekenis van de in de database opgeslagen gegevens.

De reden om, in bovenstaand voorbeeld, W1 te kiezen als waardenverzameling van attribuut anm zou kunnen zijn: lepels, vorken en messen worden uitsluitend 'los van elkaar' verkocht en ingekocht. Voor W2 zou gekozen kunnen worden als, naast deze 'losse ver- en inkoop' ook transacties plaatsvinden waarbij de combinaties lepel-

vork, lepel-vork-mes en vork-mes kunnen voorkomen, bijvoorbeeld door aanbieding van deze combinaties in één verpakking.

Over 1NF kunnen wij dus samenvattend zeggen: elk tabeltype is per definitie in 1NF, zonder daarbij te hoeven terugvallen op vage termen als 'atomair', 'ondeelbaar' e.d.

Nu enkele woorden over tweede en derde normaalvorm. Vertaald in de formele opzet van dit boek kan worden gesteld dat een tabeltype in tweede normaalvorm (2NF) is, als elk attribuut afhankelijk is van elke volledige sleutel, waarvan het geen deel uitmaakt. En een tabeltype is in derde normaalvorm (3NF) als het in 2NF is èn er geen afhankelijkheid bestaat, waarvan het linkerlid niet tot een sleutel behoort. In voorbeeld 4.3 is TT-ORDER niet in 2NF vanwege f3 tot en met f7.

Op 'hogere normaalvormen' wordt in dit boek helemaal niet ingegaan. Voor uitgebreide informatie over normaalvormen zijn verwezen naar o.a. DATE.

Gezien de 'historische betekenis' van normaalvormen is er in deze paragraaf enige aandacht aan gewijd, zonder van deze normaalvormen als zodanig overigens gebruik te maken. Hiermee wil beslist niet gezegd zijn, dat het normalisatiebegrip als zodanig niet erg zinvol zou zijn (zie ook volgende paragraaf).

5.3 ARGUMENTEN VOOR NORMALISATIE

Als argument voor normalisatie werd en wordt nog dikwijls aangevoerd, dat daarmee redundantie in opslag van gegevens zou worden vermeden. Redundante opslag van gegevens kan echter zeer zinvol zijn, afhankelijk van de wijze waarop men van de opgeslagen gegevens gebruik wil maken. Redundante opslag kan zelfs noodzakelijk zijn in verband met eisen betreffende responstijden. Ter illustratie de volgende beschouwing bij voorbeeld 5.1 (van orders, leveranciers, artikelen en assortimenten).

Stel dat een vraag naar gegevens over een order

- snel moet kunnen worden beantwoord, èn
- daarbij ook meestal de naam van de leverancier en de prijs nodig zijn.

Bij zulke responseisen zal het nuttig, zo niet noodzakelijk zijn bij elk ordertupel leveranciersnaam en prijs op te slaan. Dit kan uiteraard tot behoorlijk redundante opslag van deze twee attributen leiden. Maar dan spreken wij ook niet meer over de logische structuur, maar over de opslagstructuur. Bij figuur 1. in hoofdstuk 1 werd reeds benadrukt, dat door eisen betreffende toegang tot de gegevens

de logische structuur niet 'onverlet' overgenomen kan worden in de opslagstructuur. Er zij nogmaals op gewezen dat de logische structuur louter afhankelijk is van de betekenis van de gegevens. *Normalisatie heeft alleen betrekking op de betekenis van de gegevens.* Als gevolg van normalisatie mag wel worden verwacht dan zinloze redundantie zal worden vermeden.

5.4 VERBANDEN TUSSEN OBJECT OCCURRENCES

Tussen object-occurrences van verschillende objecten en dus tussen tupels van tabellen van verschillende tabeltypen kunnen verschillende verbanden worden onderscheiden.

Tussen tupels van tabellen T1 en T2 van twee verschillende tabeltypen TT1 en TT2 onderscheidt men de volgende verbanden:

één op één	(1 : 1)
één op veel	(1 : n)
veel op veel	(n : m)

met de volgende betekenis:

T1 T2	betekenis
1 : 1	bij elk tupel van T1 hoort precies één tupel van T2 èn omgekeerd
1 : n	bij elk tupel van T1 horen n ($n \geqq 0$) tupels van T2 èn bij elk tupel van T2 hoort precies één tupel van T1
n : m	bij elk tupel van T1 horen m ($m \geqq 0$) tupels van T2 èn bij elk tupel van T2 horen n ($n \geqq 0$) tupels van T1.

Voor de voorbeelden van deze paragraaf wordt uitgegaan van de volgende vier genormaliseerde tabeltypen:

tabeltype	voor representatie van	sleutel
TT-AF	AFdelingen	{afnr}
TT-WN	WerkNemer	{wnr}
TT-PR	PRoject	{pnr}
TT-AT	ArTikel	{atnr}

Voorbeeld 5.2

a. 1:1 verband
Elke werknemer hoort bij precies één afdeling èn elke afdeling heeft precies één werknemer in dienst.

Om dit (gekunstelde) 1:1-verband te representeren moet afnr een attribuut zijn van TT-WN èn wnr een attribuut van TT-AF. Overigens kunnen in dit geval de tabeltypen TT-AF en TT-WN door één genormaliseerd tabeltype worden vervangen. Normalisatie alleen leidt niet dwingend tot zulke samenvoeging.

b. 1:n verband
Elke werknemer hoort bij precies één afdeling èn een afdeling kan meerdere werknemers in dienst hebben.
Om dit (meer reële) 1:n-verband te representeren met genormaliseerde tabeltypen zal afnr een attribuut moeten zijn van TT-WN.

c. n:m verband
Elk artikel kan geleverd worden door meerdere afdelingen èn elke afdeling kan meerdere artikelen leveren.
Om dit n:m-verband te representeren met genormaliseerde tabeltypen zal het nodig zijn om naast de reeds genoemde tabeltypen nog een nieuw tabeltype (TT-AFAT) ter beschikking te hebben, waartoe onder meer de attributen afnr en atnr behoren. □

Voor verbanden tussen tupels van tabellen T1, T2 en T3 van drie verschillende tabeltypen TT1, TT2 en TT3 is nu verder alleen nog van belang het zogeheten 'veel op veel op veel'-verband (n1:n2:n3). De betekenis hiervan is: bij elk tupel van T1 horen meerdere tupels van T2 en T3 èn bij elk tupel van T2 horen meerdere tupels van T1 en T3 èn bij elk tupel van T3 horen meerdere tupels van T1 en T2.

Voorbeeld 5.3

n1:n2:n3 verband
Elke afdeling kan aan verschillende projecten verschillende artikelen leveren èn aan elk project kunnen door verschillende afdelingen verschillende artikelen worden geleverd èn elk artikel kan door verschillende afdelingen aan verschillende projecten worden geleverd.. Kortweg: om een individuele levering als zodanig te herkennen moeten (minstens) bekend zijn: afdeling, project en artikel.

Om dit verband te representeren met genormaliseerde tabeltypen zal het nodig zijn een nieuw tabeltype ter beschikking te hebben, waartoe onder meer de attributen afnr, pnr en atnr behoren. □

OPGAVEN

5.1 (bij voorbeelden 4.3 en 5.1)
Uitgegaan wordt van K-ORDER als in voorbeeld 4.3. Verder worden vier los van elkaar staande specificaties gegeven als hieronder sub a tot en met d.

a. Verder gelden f1 èn f3 tot en met f7.
b. Verder gelden f1 tot en met f7 èn {arc} → {hoev}.
c. Verder gelden f1 tot en met f5 èn f7 en {lnr} → {pr}.
d. Verder gelden f1 tot en met f5 èn f7 èn de tupelconstraint C1 en de tabelconstraint C2 met:

C1(t) := [t(arc) = A16 ⇒ t(pr) = 83] èn
C2(D) := [#({t(lnr) | t ∈ D}) ≤ 10].

Geef, voor elk van de gevallen a, b, c, d apart, hoe de gegevensstructuur kan worden weergegeven met genormaliseerde tabeltypen.

5.2 Gegeven zijn onderstaande tabeltypen TT-PAT (patiënten), TT-OPN (opnamen) en TT-SP (specialisten). TT-PAT met K-PAT als kleinste object-karakterisering en de afhankelijkheid f1.

K-PAT =	Toelichting
{(pnr,{1...1000000}),	nummer
(pnm,charstring 25),	naam
(padr,charstring 20),	adres
(pwpl,charstring 20),	woonplaats
(gbd,integer)	geboortedatum
(gesl,{m,v})}	geslacht

f1: {pnr} → dom(K-PAT)

TT-OPN met K-OPN als kleinste object-karakterisering en de afhankelijkheden f2-f6.

K-OPN =	Toelichting
{(pnr,{1...1000000}),	patiëntnummer
(pnm,charstring 25),	patiëntnaam
(gbd,integer)	geboortedatum
(indat,{700101...991231}),	opnamedatum
(uitdat,{700101...991231}),	ontslagdatum
(opnr,charstring 25),	reden opname
(spnr,{1...2000}),	nummer specialist
(snm,charstring 25),	naam specialist
(anr,{1...200}),	afdelingsnummer
(rnr,{1...1000}),	nummer van verpleegruimte

f2: {pnr,indat} → dom(K-OPN).
f3: {pnr} ↔ {pnm}
f4: {pnm} → {gbd}
f5: {spnr} → {snm}
f6: {rnr} → {anr}

TT-SP met K-SP als kleinste object-karakterisering en de afhankelijkheid f7.

K-SP =	Toelichting
{(spnr,{1...2000}),	nummer
(snm,charstring 25),	naam
(sadr,charstring 20),	adres
(swpl,charstring 20),	woonplaats
(anr, {1...200}),	afdelingsnummer
(telefnr,integer)}	telefoonnummer

f7: {spnr} → dom(K-SP)

Geef met behulp van genormaliseerde tabeltypen de informatie van bovenstaande tabeltypen weer.

5.3 (uitgangspunt bij deze opgave zijn de vier tabeltypen, genoemd aan het begin van paragraaf 5.4)
Wat is nodig om elk van de onderstaande specificaties te representeren?

a. Elk project beschikt over meerdere werknemers en elke werknemer is aan precies één project toegevoegd.
b. Elke afdeling kan verschillende artikelen aan verschillende projecten leveren èn elk artikel wordt aan elk project door precies één leverancier geleverd.

5.4 Welke verbanden (in de zin van paragraaf 5.4) liggen opgesloten in voorbeeld 5.1?

6 DATABASETYPE; SUBSET REQUIREMENTS

6.1 DATABASETYPE

In het vorige hoofdstuk werd al enigszins gewag gemaakt van mogelijke verbanden tussen tupels van tabellen van verschillende tabeltypen (bijvoorbeeld tussen tupels van tabellen van de tabeltypen TT-ORDER-N, TT-LEV-N, TT-ART-N en TT-ASS-N van voorbeeld 5.1). In 'bestands'- en 'recordtermen' gaat het om mogelijke verbanden tussen records van verschillende bestanden. Wij zijn dus geïnteresseerd, niet in één bestand, maar in een verzameling van bestanden. Zo'n verzameling van bestanden wordt 'losweg' aangeduid met de term 'database'. In tegenstelling tot de begrippen 'bestand (file)' en 'record' heeft het begrip 'database' nog geen plaats gekregen in programmeertalen als Pascal. Wel komt het voor in de zogeheten DDL (Data Description Language) van verschillende database management systemen. In het algemeen kan men zeggen dat het in deze systemen gebruikte begrip database een bijzonder geval is van het hier te definiëren begrip databasetype.

In deze definitie zullen wij gebruik maken van het begrip 'verwante' functies.

Twee functies zijn *verwant*, als zij niet leeg zijn en hetzelfde domein hebben. Als V een verzameling verwante functies is, noemen wij het gemeenschappelijke domein van deze functies *het domein van V* (notatie: dom(V)).

Nu volgt de definitie van databasetype.

Definitie 6.1
(niet-formeel) Een *databasetype* DT is een niet-lege verzameling verwante functies, waarvoor geldt dat per element van het domein van DT de verzameling van alle beelden een tabeltype is.
(formeel)
DT is een *databasetype* $\stackrel{D}{\equiv}$ DT $\neq \emptyset \wedge$

DT is een verzameling verwante functies èn

$\forall d \in \mathrm{dom}(DT)$ [$\{f(d) \mid f \in DT\}$ is een tabeltype]. □

Een element van een databasetype noemen wij een *database* (vergelijk: tupel is element van tupeltype; tabel is element van tabeltype).

Voorbeeld 6.1

Het databasetype DT-HANDEL-1 is als volgt gedefinieerd:

```
DT-HANDEL-1 = {H| H is een functie ∧
      dom(H) = {lev,art,ass,order} ∧
      H(lev) ∈ TT-LEV-N ∧ H(art) ∈ TT-ART-N ∧
      H(ass) ∈ TT-ASS-N ∧ H(order) ∈ TT-ORDER-N},
```

waarbij TT-LEV-N enz. als in voorbeeld 5.1.
De hieronder weergegeven database H1 is een van de (zeer vele) elementen van het databasetype DT-HANDEL-1.

```
H1 =
   {(lev,{{(lnr,5),(lnm,brakes),(ladr,pad 5),(lwpl,eindhoven)},
          {(lnr,7),(lnm,nemmer),(ladr,pad 9),(lwpl,nuenen)},
          {(lnr,23),(lnm,frein),(ladr,badweg 16),(lwpl,eindhoven)}}),

    (art,{{(arc,15),(anm,microz),(bestadr,pad 7)},
          {(arc,77),(anm,minib),(bestadr,ringbaan 5)},
          {(arc,63),(anm,printer),(bestadr,badweg 5)},
          {(arc,50),(anm,visdisplay),(bestadr,pad 5)}}),

    (ass,{{(lnr,5),(arc,77),(pr,510)},
          {(lnr,23),(arc,63),(pr,210)},
          {(lnr,5),(arc,50),(pr,53)},
          {(lnr,23),(arc,15),(pr,62)},
          {(lnr,23),(arc,50),(pr,55)}}),

    (order,{{(onr,12),(dat,811206),(lnr,5),(arc,77),(hoev,15)},
            {(onr,26),(dat,820211),(lnr,23),(arc,15),(hoev,80)}})}.
```

H1(lev) enz. zien er in tabelvorm als volgt uit:

H1(lev) =

lnr	lnm	ladr	lwpl
5	brakes	pad 5	eindhoven
7	nemmer	pad 9	nuenen
23	frein	badweg 16	eindhoven

H1(art) =

arc	anm	bestadr
15	microz	pad 7
77	minib	ringbaan 5
63	printer	badweg 5
50	visdisplay	pad 5

H1(ass) =

lnr	arc	pr
5	77	510
23	63	210
5	50	53
23	15	62
23	50	55

H1(order) =

onr	dat	lnr	arc	hoev
12	811206	5	77	15
26	820211	23	15	80

In 'bestandstermen' zal elke database van DT-HANDEL-1 vier bestanden bevatten, en wel een leveranciersbestand H(lev), een artikelbestand H(art), een assortimentsbestand H(ass) en een orderbestand H(order). De inhoud van deze vier bestanden is verschillend per database. In bovengenoemd element H1 bestaat het bestand H1(lev) uit drie tupels (records), het bestand H1(art) uit vier tupels, het bestand H1(ass) uit vijf tupels en het bestand H1(order) uit twee tupels. □

Dom(DT), het gemeenschappelijke domein van de databases van een databasetype DT, noemen wij de *objectverzameling* van DT. Een element van de objectverzameling van DT noemen wij een *object* van DT. Zo is {lev,art,ass,order} de objectverzameling van DT-HANDEL-1 (voorbeeld 6.1).

Onder de *attributen van een object* verstaan wij de attributen van het tabeltype dat bij het object behoort.

Evenals bij een tupeltype, geldt ook voor een databasetype DT, dat er verzamelingsfuncties bestaan, zó dat voor elk van deze verzamelingsfuncties geldt dat DT een deelverzameling is van het produkt. Zo'n verzamelingsfunctie noemen wij een *databasekarakterisering* (vergelijk: objectkarakterisering bij een tupeltype).

Voorbeeld 6.2

De databasekarakterisering FDT-HANDEL is als volgt gedefinieerd:

FDT-HANDEL = { (lev,TT-LEV-N),
(art,TT-ART-N),
(ass,TT-ASS-N),
(order,TT-ORDER-N)},

waarbij de tabeltypen TT-LEV-N enz. als in voorbeeld 5.1.

Het is direct in te zien dat DT-HANDEL-1 van voorbeeld 6.1 een deelverzameling is van Π(FDT-HANDEL), zelfs DT-HANDEL-1 =

Π(FDT-HANDEL). Dus FDT-HANDEL is een databasekarakterisering van DT-HANDEL-1. □

Bij de behandeling van tupeltype in hoofdstuk 3 zagen wij, dat in het algemeen een tupeltype slechts een relatief kleine deelverzameling is van het produkt van de objectkarakterisering, die men als uitgangspunt neemt. Dit wordt, zoals bekend, veroorzaakt door de 'inperking' vanwege tupelconstraints.

Ook het aantal mogelijke waarden van een databasetype zal beperkt kunnen worden en wel door zogeheten databaseconstraints.

Een *databaseconstraint* is een predikaat over het produkt van een databasekarakterisering.

Voorbeeld 6.3

Over Π(FDT-HANDEL) van voorbeeld 6.2 worden de constraints C1 tot en met C4 gedefinieerd. DB $\in$ Π(FDT-HANDEL).

C1(DB) := [#(DB(lev)) ≤ #(DB(ass))].

C2(DB) := [∀t∈DB(order) [∃u∈DB(lev) [t(lnr) = u(lnr)] ∧
∃v∈DB(art) [t(arc) = v(arc)]]].

C3(DB) := [∀t∈DB(order) [∃u∈DB(ass)
[t(lnr) = u(lnr) ∧ t(arc) = u(arc)]]].

C4(DB) := [∀t∈DB(art) [t(bestadr) = Veem 6 ⇒
sum2c({(u,u(hoev)) | u ∈ DB(order) ∧ t(arc) =
u(arc)}) ≤ 1000]. □

Opmerking
De functie 2c levert de som van de tweede coördinaten van de paren (u,u(hoev)). Er zij verder opgemerkt, dat sum({u(hoev) |.... }), waarbij de functie sum de som levert van de getallen u(hoev), in het algemeen niet gelijk zal zijn aan sum2c. Als namelijk hoev in twee tupels van DB(order) dezelfde waarde heeft, dan zal deze waarde in {u(hoev) |.... } ook maar eenmaal voorkomen.

In woorden (met uiteraard het nodige risico voor dubbelzinnigheden) luiden bovenstaande constraints als volgt:

C1: Het aantal leveranciers(tupels) mag niet groter zijn dan het aantal assortimenten(tupels).

C2: Bij elk ordertupel moet een leverancierstupel respectievelijk artikeltupel voorkomen met dezelfde waarde voor het attribuut lnr respectievelijk arc. Met andere woorden: geen ordertupel

zonder dat het bijbehorende leverancierstupel respectievelijk artikeltupel in de database aanwezig is.

C3: Bij elk ordertupel moet een assortimenttupel voorkomen met dezelfde waarde voor de attributen lnr en arc.

C4: Van elk artikel, met besteladres Veem 6, mag het totaal aantal stuks in bestelling niet groter dan 1000 zijn. □

De definitie van een databasetype met behulp van een databasekarakterisering en een (karakteriserende) databaseconstraint vindt op analoge wijze plaats als de definitie van een tupeltype met behulp van een objectkarakterisering en een (karakteriserende) tupelconstraint.

Voorbeeld 6.4

Met behulp van FDT-HANDEL (voorbeeld 6.2) en de karakteriserende databaseconstraint $C = C1 \wedge C2 \wedge C3 \wedge C4$ (voorbeeld 6.3) over Π(FDT-HANDEL) kan databasetype DT-HANDEL-2 worden gedefinieerd:

$$\text{DT-HANDEL-2} = \{DB \mid DB \in \Pi(\text{FDT-HANDEL}) \wedge C(DB)\}.$$ □

Databasetype DT-HANDEL-2 is een echte deelverzameling van Π(FDT-HANDEL), terwijl, zoals we reeds zagen, databasetype DT-HANDEL-1 gelijk is aan Π(FDT-HANDEL) (voorbeeld 6.2). DT-HANDEL-1 is een bijzonder geval van een databasetype, namelijk een databasetype waarvoor geen enkele databaseconstraint geldt (dus formeel: C = true). Dit bijzondere geval is het gebruikelijke databasetype in de bestaande database management systemen.

Tenslotte merken we nog op dat analoog aan de kleinste objectkarakterisering van een tupeltype, ook kan worden gesproken van de *kleinste* databasekarakterisering van een databasetype.

6.2 SUBSET REQUIREMENTS

In het vorige hoofdstuk was sprake van het behoud van informatie bij overgang van ongenormaliseerde tabeltypen naar genormaliseerde tabeltypen. Daartoe werden bepaalde attributen en afhankelijkheden 'ondergebracht' in andere reeds bestaande of nieuw te definiëren tabeltypen. Eén kwestie is daarbij echter onbesproken gebleven, namelijk hoe op formele wijze het verband wordt behouden tussen de 'verplaatste' attributen en de 'achtergebleven' attributen. Wordt dit verband niet formeel vastgelegd, dan gaat informatie verloren. Immers, hoe zal het systeem (zonder extra maatregelen onzerzijds, zoals bijvoorbeeld via een toepassingsprogramma) kunnen weten dat

het attribuut 'lnr' van tabeltype TT-ORDER-N iets of zelfs heel veel te maken heeft met het attribuut 'lnr' van tabeltype TT-LEV-N. In het vorige hoofdstuk kozen wij voor gelijke attribuutnamen als een 'voorlopige' oplossing. Sommige auteurs zien dit ook inderdaad als de juiste oplossing, dus gelijke attribuutnamen zijn alleen toegestaan als deze worden toegekend aan gelijke attributen en omgekeerd. Nog afgezien van de vraag, wat men in zo'n geval precies dient te verstaan onder *gelijke* attributen, is met juist genoemde oplossing het behoud van informatie niet gegarandeerd. Immers, zolang men 'er niet van op aan kan' dat bijvoorbeeld alle leverancierstupels, die in de ongenormaliseerde opzet als 'deeltupels' in een element van het tabeltype TT-ORDER voorkomen, in de genormaliseerde opzet als tupels in een element van het tabeltype TT-LEV-N zullen voorkomen, kan bezwaarlijk worden gesproken van behoud van informatie. Het zal dan ook nodig zijn formeel de nodige verbanden te leggen tussen tupels van verschillende 'componenten' van een databasetype. Deze verbanden zullen wij leggen met behulp van speciale databaseconstraints, de zogeheten *subset requirements*.

Alvorens nu een en ander formeel vast te leggen, zullen wij eerst aan de hand van een voorbeeld de zaak informeel inleiden.

In de database H1 van DT-HANDEL-1 in voorbeeld 6.1 komen in H1(ass) van het attribuut 'lnr' alleen waarden voor die ook voorkomen in H1(lev) als waarden van het attribuut 'lnr'. Het omgekeerde blijkt overigens niet te gelden. Waar wij nu naartoe willen is de volgende speciale constraint over de databases van Π(FDT-HANDEL) van voorbeeld 6.2: alleen die databases H zullen worden toegelaten, waarvoor geldt dat elke lnr-waarde in H(ass) ook voorkomt als lnr-waarde in H(lev). Met andere woorden: bij elk assortimentstupel moet 'te zelfder tijd' het 'bijbehorende' leverancierstupel beschikbaar zijn. In voorbeeld 6.3 zijn de constraints C2 en C3 soortgelijke voorbeelden, bij C3 is zelfs sprake van gelijke waarden voor twee attributen, lnr en arc.

Het is goed denkbaar dat voor 'overeenkomstige' attributen van verschillende objecten verschillende namen worden gebruikt.

Voorbeeld 6.5

Stel dat FDT-HANDEL-2 gelijk is aan FDT-HANDEL van voorbeeld 6.2 op de volgende 'vervangingen' na:

In TT-LEV-N:	'lno'	i.p.v.	'lnr'.	
In TT-ASS-N:	'anr'	i.p.v.	'arc'.	
In TT-ASS-N:	'levnr'	i.p.v.	'lnr'	èn
	'artk'	i.p.v.	'arc'.	

Over Π(FDT-HANDEL-2) willen wij nu 'dezelfde' constraints definiëren als C1, C2, C3 en C4 van voorbeeld 6.3. C1 blijft dan ongewijzigd, terwijl C2', C3', C4' komen in plaats van C2, C3, C4.

$$C2'(DB) = [\forall t \in DB(order)[\exists u \in DB(lev)[t(lnr) = u(lno)] \wedge \exists v \in DB(art)[t(arc) = v(anr)]]].$$

$$C3'(DB) = [\forall t \in DB(order)[\exists u \in DB(ass)[t(lnr) = u(levnr) \wedge t(arc) = u(artk)]]].$$

$$C4'(DB) = [\forall t \in DB(art)[t(bestadr) = \text{Veem 6} \Rightarrow sum2c(\{(u,u(hoev)) \mid u \in DB(order) \wedge t(anr) = u(arc)\}) \leq 1000]].$$ □

In voorbeeld 6.5 is sprake van zogeheten attributentransformatie, kortweg *attribtrafo*.

Een *attribtrafo* is een functie, die twee verzamelingen van attributen éénéén-duidig op elkaar afbeeldt. Zo zijn de functies f1, f2 en f3 met f1 = {(lnr,lnr)}, f2 = {(lnr,lno)} en f3 = {(lnr,levnr), (arc,artk)} attribtrafo's.
Meer *formeel*: een *attribtrafo* is een functie, waarvoor geldt:

- dom(f) = A1; rng(f) = A2;
- A1 en A2 zijn attributenverzamelingen
- |A1| = |A2|.

Wij spreken ook wel van een attribtrafo van A1 naar A2.

Wij zijn nu toe aan de definiëring van het begrip subset requirements. Hierbij dient goed in het oog te worden gehouden dat, als DK een databasekarakterisering is en OB een object van DK, de uitdrukking DK(OB) een tabeltype voorstelt. Het is dus zinvol te spreken van de attributenverzameling van DK(OB). Zo is (zie voorbeeld 6.2) de attributenverzameling van FDT-HANDEL(art) gelijk aan {arc,anm,bestadr} (zie voorbeeld 5.1).

Definitie 6.2

Zij: DK: een databasekarakterisering,
OB1,OB2: twee objecten van DK,
A1,A2: een deelverzameling van de attributenverzameling van DK(OB1),DK(OB2),
f: een attribtrafo van A1 naar A2.

(niet-formeel) De databaseconstraint C is een *subset requirement vanuit A1 van OB1 naar A2 van OB2 met f*, als voor elke database van Π(DK), die aan C voldoet, geldt dat bij elk tupel van OB1 een tupel van OB2 voorkomt, zó dat A2 en A1 gelijke waarden hebben voor elk paar attributen, dat volgens f bij elkaar hoort.
(formeel) C is een *subset requirement vanuit A1 van OB1 naar A2 van OB2 met f* $\stackrel{D}{\equiv}$

$$C(DB) = [\forall t \in DB(OB1) // \overline{A1} [\{(f(a1),t(a1)) \mid a1 \in A1\} \in DB(OB2) // \overline{A2}]].$$ □

Voor subset requirement zullen wij verder de afkorting ssr gebruiken.

Voorbeeld 6.6

C2 van voorbeeld 6.3 bevat twee ssr's, namelijk

1. vanuit {lnr} van order naar {lnr} van lev met f = {(lnr,lnr)},
2. vanuit {arc} van order naar {arc} van art met f = {(arc,arc)}.

C3 van voorbeeld 6.3 is een ssr

vanuit {lnr,arc} van order naar {lnr,arc} van ass met

f = {(lnr,lnr),(arc,arc)}.

C3' van voorbeeld 6.5 is een ssr

vanuit {lnr,arc} van order naar {levnr,artk} van ass met

f = {(lnr,levnr),(arc,artk)}. □

Opmerking bij definitie 6.2
Als A2 een sleutel is van DK(OB2), dan noemt men A1 een *referentiesleutel (foreign key) ten opzichte van A2.*

6.3 GENORMALISEERD DATABASETYPE

In het vorige hoofdstuk hebben wij het begrip genormaliseerd tabeltype ingevoerd. Met behulp hiervan kunnen wij de definitie geven van een zogeheten genormaliseerd databasetype.

Definitie 6.3
(niet-formeel) Een databasetype DT is *genormaliseerd* als de tabeltypen die bij de objecten horen alle genormaliseerd zijn.
(formeel) Een databasetype DT is *genormaliseerd*

$\stackrel{D}{\equiv}$ [∀OB ∈ dom(DT) [het tabeltype van OB is genormaliseerd]]. □

Voorbeeld 6.7

Databasetype DT-HANDEL-1 van voorbeeld 6-1 is genormaliseerd, want de bij de objecten lev, art, ass en order behorende tabeltypen TT-LEV-N, TT-ART-N, TT-ASS-N en TT-ORDER-N zijn genormaliseerd (zie voorbeeld 5.1).

Zo is ook databasetype DT-HANDEL-2 van voorbeeld 6.4 genormaliseerd. □

Bij overgang van ongenormaliseerd databasetype naar genormaliseerd databasetype, is de normalisatie geen voldoende voorwaarde voor behoud van informatie. Hiervoor zullen nog de nodige voorzieningen moeten worden getroffen met behulp van ssr's, en wel volgens onderstaand voorschrift.

Als databasetype DTN genormaliseerd is en minstens dezelfde informatie bevat als databasetype DT, dan moet aan de volgende voorwaarden zijn voldaan:
- DTN bevat alle attributen en afhankelijkheden van DT.
- Voor elke afhankelijkheid, die de normalisatie van DT verstoort, is in DTN een ssr gedefinieerd.

Voorbeeld 6.8

Stel DT-HANDEL-0 is het databasetype met 'order' als enig object en TT-ORDER van voorbeeld 4.3 als tabeltype behorende bij dit object. Dan is duidelijk dat DT-HANDEL-0 een ongenormaliseerd databasetype is.

DT-HANDEL-1 van voorbeeld 6.1 is weliswaar een genormaliseerd databasetype, maar 'omvat' niet de informatie van DT-HANDEL-0 vanwege het ontbreken van ssr's. DT-HANDEL-2 van voorbeeld 6.4 is echter wel een genormaliseerd databasetype, dat de geschikte ssr's bevat om de informatie van DT-HANDEL-0 te omvatten. (Men ga dit na; zie ook opgave 6.2b3). □

OPGAVEN

6.1 (H1 en DT-HANDEL-1 als in voorbeeld 6.1;
DT-HANDEL-2 als in voorbeeld 6.4;
FDT-HANDEL als in voorbeeld 6.2).

a. Behoort database H1 tot databasetype DT-HANDEL-2?

b. Gegeven de databases H2, H3, H4, H5 van Π(FDT-HANDEL) met onderstaande definities:

H2: $H2 \restriction \{lev, art, order\} = H1 \restriction \{lev, art, order\}$ èn $H2(ass) = \emptyset$

H3: $H3(lev) = H3(art) = H3(ass) = H3(order) = \emptyset$

H4: $H4 \restriction \{lev, art, ass\} = H1 \restriction \{lev, art, ass\}$ èn $H4(order) = \emptyset$

H5: $H5(lev) = H1(lev) \cup \{\{(lnr, 18), (lnm, frein), (ladr, plein\ 17), (lwpl, geldrop)\}\}$.

Ga voor elk van deze vier databases na of zij behoren tot databasetype DF-HANDEL-2.

c. Geef de volgende databaseconstraints C5 en C6 over Π(FDT-HANDEL) in woorden weer:

$$C5(DB) := [\#(\{t \mid t \in DB(order) \wedge 800101 \leq t(dat) \leq 800601 \wedge \exists u \in DB(lev)[t(lnr) = u(lnr) \wedge u(lwpl) = eindhoven]\}) \leq 100].$$

$$C6(DB) := [\forall t \in DB(ass)[t(pr) > 500 \Rightarrow \forall u \in DB(order)[(u(lnr) = t(lnr) \wedge u(arc) = t(arc)) \Rightarrow u(hoev) < 25]]].$$

d. Geef de volgende databaseconstraints over Π(FDT-HANDEL) formeel weer.
   d1. Het aantal orders in januari 1981 voor besteladres 'Mag. 8' mag niet groter zijn dan 20.
   d2. Orders in januari 1981 kunnen niet worden afgeleverd op besteladres 'Veem 13'.

6.2 (bij voorbeeld 6.3)
a. Welke van onderstaande databaseconstraints Ca, Cb, Cc, is equivalent met C3?

$$Ca(DB) := [\forall t \in DB(order)[\exists u \in DB(ass)[t/\overline{\{lnr, arc\}} = u/\overline{\{lnr, arc\}}]]].$$

$$Cb(DB) := [\forall t \in DB(ass)[\exists u \in DB(order)[t(lnr) = u(lnr) \wedge t(arc) = u(arc)]]].$$

$$Cc(DB) := [\exists t \in DB(order)[\forall u \in DB(ass)[t(lnr) = u(lnr) \wedge t(arc) = u(arc)]]].$$

b. Gegeven de databaseconstraint C7:

$$C7[DB] := [\forall t \in DB(ass)[\exists u \in DB(lev)[u(lnr) = t(lnr)] \wedge \exists v \in DB(art)[v(arc) = t(arc)]]].$$

   b1. Ga na dat C7 een ssr is.
   b2. Geldt: $(C2 \wedge C3)(DB) \Rightarrow C5(DB)$?
   b3. Gegeven DT-HANDEL-3 = $\{DB \mid DB \in \Pi(FDT\text{-}HANDEL \wedge (C2 \wedge C7)(DB)\}$.
   Is DT-HANDEL-3 een databasetype met minstens dezelfde informatie als DT-HANDEL-0 (voor voorbeeld 6.8)?

6.3 Een verzekeringsmaatschappij wil een database bouwen ten behoeve van een efficiënte informatievoorziening bij het afsluiten en behandelen van polissen. Daartoe zijn de volgende specificaties opgesteld.
De verschillende verzekeringssoorten zijn ingedeeld in (elkaar niet-overlappende) klassen. Voorbeelden van klassen zijn: de klasse van autoverzekeringen, de klasse van ziektekostenverzekeringen. Voorbeelden van verzekeringssoorten binnen de klasse van autoverzekeringen zijn: all-risk verzekering, W.A.-verzekering. Per verzekeringssoort zullen in het algemeen vele polissen worden afgesloten.
Elke klasse valt onder de verantwoordelijkheid van een of meer medewerkers. Deze verantwoordelijkheid houdt in dat informatie over de desbetreffende klasse aan klanten kan worden verstrekt en dat polissen voor verzekeringssoorten van de desbetreffende klasse mogen worden afgesloten.
Per polis is precies één medewerker verantwoordelijk voor afsluiting en behandeling van deze polis.
De mate van deskundigheid van een medewerker kan per klasse verschillend zijn.
De volgende gegevens moeten via de database beschikbaar kunnen komen (met in hoofdletters: voorstel voor eventueel benodigde namen).

Per klasse (KL):
klassecode (KC, uniek), klassebeschrijving (KB), naam (MNM) van elke verantwoordelijke medewerker en diens mate van deskundigheid (DSK) voor de desbetreffende klasse.
Per verzekeringssoort (VERZ):
KC (van klasse), subcode (SUBC, uniek binnen KC), beschrijving (VB), aantal afgesloten polissen in elk van de jaren 1977, 1978, 1979 (APV7, APV8, APV9).
Per nedewerker (MEDEW):
naam (MNM, uniek), kantooradres (KADR), kantoorwoonplaats (KWPL), privé-adres (PADR), privé-woonplaats (PWPL), telefoonnummer kantoor (TELK), telefoonnummer privé (TELP), aantal afgesloten polissen in elk van de jaren 1977, 1978, 1979 (APM7, APM8, APM9).
Per klant (KLANT)
klantnummer (KLNR, uniek), naam (KNM), adres (KADR), woonplaats (KWPL), aantal geldige polissen (AGP).
Per informatieverstrekking (INFVSTR):
KC (van klasse), naam (IMNM) van medewerker, mate van deskundigheid (IDSK) van medewerker voor de klasse, hoogst mogelijke mate van deskundigheid (HDSK) binnen de klasse, klantnummer (IKLNR) van een bestaande klant, datum (DAT), manier van communicatie (MC, bijvoorbeeld per brief of telefoon).

Voor de unieke identificatie van een informatieverstrekking is de combinatie (KC, IMNM, IKLNR, DAT) nodig.

Per polis (POL):

klassecode (PKC) van klasse, subcode (PSUBC) van verzekeringssoort, naam (PMNM) van medewerker, klantnummer (PKLNR) van bestaande klant, volgnummer (VNR, nodig omdat een klant van eenzelfde verzekeringssoort meer dan een polis kan afsluiten), premie (PR), afsluitingsdatum (PDAT), looptijd (LPT, na het verstrijken van de looptijd is de polis ongeldig).

Ontwerp een genormaliseerd databasetype voor bovenstaande specificaties. Zonodig zelf hiaten in de semantiek van de gegevens aanvullen.

7 UITGEBREID VOORBEELD VAN EEN GENORMALISEERD DATABASETYPE (ziekenhuis-database)

Hieronder volgt de definitie van een genormaliseerd databasetype, waarbij in opeenvolgende delen van voorgaande type-definities gebruik wordt gemaakt. In de kolom toelichtingen wordt de betekenis van de onderdelen toegelicht.

Bij deze definitie zullen wij gebruik maken van onderstaande syntax:

1. subrange, array en record als in Pascal.

2. TT = object
   kar OK
   tuc C(t)
   tac C(D)
   keys SL
   endobject

   voor de definitie van tabeltype TT, met objectkarakterisering OK èn

   facultatief { tupelconstraint C(t) (t ∈ Π(OK)), èn
   tabelconstraint C(D) (D ∈ P(MT); MT kleinste karakteriserend tupeltype), èn
   sleutelverzameling SL.

3. DT = database
   kar DK
   ssr (O1.A1,O2.A2,f)
   dac C(DB)
   enddatabase

   voor de definitie van databasetype DT, met databasekarakterisering DK èn de ssr's vanuit A1 van O1 naar A2 van O2 met f (id: identieke afbeelding) èn databaseconstraint C(DB) (DB ∈ Π(DK)).

```
Type  nr = {1...100000};
      string = array [1...25] of char;
      adres = record straat : string;
                     hnr : nr;
              endrecord;
      hoev = {1...10000};
      datum = {700101...991231};

     patiënt = object
               kar pnr : nr;
                   pnm : string;
                   padr : string;
                   pwpl : string;
                   gbdat : {8800101...9991231};
                   blgr : {O,A,B,AB};
                   rhf : {+,-};
                   gesl : {m,v}
               tac #{t| t ∈ D ∧ t(pwpl) =
                   eindhoven} ≧ 0.5(#(D))
               keys {{pnr}}
               endobject;
```

Toelichting

Definitie van herhaaldelijk te gebruiken typen; bij de string is ook de spatie als character toegestaan.

patiëntnummer
naam
adres
woonplaats
geboortedatum
bloedgroep
rhesusfactor
geslacht

	Toelichting
verplk = object	verpleegkundige
kar vknr : nr;	verpleegkundenummer
vknm : string;	naam
vkadr : string;	adres
vkwpl : string;	woonplaats
vkanr : nr	afdelingsnummer
tuc t(vkanr) = 9 ⇒	
t(vkwpl) = eindhoven	
tac #({t| t ∈ D ∧ t(vkanr) = 9}) > 3	
keys {{vknr}}	
endonject;	
spec = object	specialist
kar snr : nr;	specialistnummer
snm : string;	naam
sadr : string;	adres
swpl : string;	woonplaats
spanr : nr	afdelingsnummer
ab : hoev	aantal beschikbare bedden
ind : {0,1}	0: niet in dienst 1: wel in dienst
tac #({t| t ∈ D ∧ t(spanr) = 9 ∧	
t(swpl) = eindhoven}) ≥ 2	

	Toelichting
keys {{snr},{snm,sadr,swpl}}	
endobject;	
afd = object	afdeling
kar anr : nr;	afdelingsnummer
anm : string;	naam
vknr : nr;	nummer hoofdverpleegkundige
snr : nr	nummer hoofdspecialist
keys {{anr},{anm}}	
endobject;	
verplr = object	verpleegruimte
kar vnr : nr;	nummer van verpleegruimte
vanr : nr;	afdelingsnummer
ab : hoev	aantal beschikbare bedden
tuc t(vanr) = 9 ⇒ (t(ab) = 2 ∧	
t(vnr) < 5)	
keys {{vnr}}	
endobject;	
opname = object	opgenomen patiënt
kar pnr : nr;	nummer opgenomen patiënt
opnr : string;	opnamereden
vnr : nr;	verpleegruimtenummer

	Toelichting
indat : datum;	opnamedatum
uitdat : datum;	vertrekdatum
snr : nr	nummer van specialist die verantwoordelijk is voor opname
tuc t(uitdat) > t(indat);	
t(opnr) = informaritis ⇒	
(t(vnr) < 5 ∧	
ad((t(indat),t(uitdat))) > 4)	ad is de functie die aan een tweetal data het absolute verschil (in dagen) toevoegt
keys {{pnr,indat}}	
endobject;	
behandeling = object	
kar bcd : string;	behandelcode
bnm : string;	naam
bsrt : string;	behandelsoort
btar : hoev	tarief
tuc t(bsrt) = KNO ⇒ t(btar) ≦ 75	
tac max({t(btar) | t ∈ D}) ≦	
10 * min({t(btar) | t ∈ D})	
keys {{bcd},{bnm}}	
endobject;	

	Toelichting
patbehandeling = object	
kar pnr : nr;	patiëntnummer
bcd : string;	behandelcode
snr : nr;	specialistnummer
indat : datum;	opnamedatum
dat : datum;	datum behandeling
duur : hoev;	duur in minuten
behr : string	behandelruimte
tuc t(bcd) = KN08 ⇒ (t(snr) ∈ {12,14} ∧	
t(behr) ≠ A12); t(dat) ≧ t(indat)	
tac ∀t1,t2∈D[t1 = t2 v	
t1(pnr) ≠ t2(pnr) v	
t1(bcd) ≠ KN02 v t2(bcd) ≠ KN02 v	
ad((t1(dat),t2(dat))) > 4]	voor definitie ad zie bij 'opname'
keys {[pnr,bcd,dat}}	
endobject;	
medicijn = object	
kar mcd : string;	medicijncode
mnm : string;	naam
msrt : string;	soort
gvc : hoev	gevarencode

	Toelichting
tuc t(mstr) = X12 ⇔ t(gvc) = 10	
tac ∀t∈D[t(gvc) = 12 ⇒	
∃u∈D[u(msrt) =	
t(msrt) ∧ u(gvc) = 6]].	
keys {{mcd},{mnm}}	
endobject;	
medicijnverst = object	medicijnverstrekking
kar pnr : nr;	patiëntnummer
mcd : string;	medicijncode
snr : nr;	nummer van voorschrijvende specialist
dat : datum;	datum van ingang
duur: hoev;	aantal dagen
fd : hoev;	frequentie per dag
am : hoev	aantal stuks per keer
tuc t(mcd) = M12 ⇒ (t(fd) ≦ 2 ∧	
t(am) ≦ 4 ∧ t(fd) * t(am) ≦ 6)	
tac ∀t1,t2∈D[(t1(pnr) =	
t2(pnr) ∧ t1(mcd) = t2(mcd) = M12) ⇒	
ad((t1(dat),t2(dat))) ≦ 15]	voor definitie ad zie bij 'opname'
keys {{pnr,mcd,dat}}	
endobject;	

```
ziekenhuis = database
            kar p : patiënt;
                vk : verplk;
                sp : spec;
                afd : afd;
                vr : verplr;
                opn : opname;
                beh : behandeling;
                pb : patbehandeling;
                m : medicijn;
                mv : medicijnverst
            ssr (afd.{vknr},vk.{vknr},id);
                (afd.{snr},sp.{snr},id);
                (vk.{vkanr},afd.{anr}.{(vkanr,anr)});
                (sp.{spanr},afd.{anr},{(spanr,anr)});
                (vr.{vanr},afd.{anr},{(vanr,anr)});
                (opn.{pnr},p.{pnr},id);
                (opn.{vnr},vr.{vnr},id);
                (opn.{snr},sp.{snr},id);
                (pb.{pnr,indat},opn.{pnr,indat},id);
                (pb.{pnr},p.{pnr},id);
```

Toelichting

```
        (pb.{snr},sp.{snr},id);
        (pb.{bcd},beh.{bcd},id);
        (mv.{pnr},p.{pnr},id);
        (mv.{mcd},m.{mcd},id);
        (mv.{snr},sp.{snr},id)
    dac ∀t∈DB(opn)[(t(opnr) = informaritis ∧
        ∃u∈DB(p)[(u(pnr) = t(pnr) ∧
        u(gbd) < 9000101]) ⇒
        ad((t(uitdat,t(indat))) > 15]];
        ∀t∈DB(mv)[∃u∈DB(opn)[u(pnr) = t(pnr)
        ∧ u(indat) ≦ t(dat) ≦ u(uitdat)]].
    enddatabase
endtype
```

Toelichting

voor definitie ad zie bij 'opname' (bij de regel `ad((t(uitdat,t(indat))) > 15]];`)

OPGAVEN

7.1 Geef alle tupelconstraints (tuc's), tabelconstraints (tac's) en databaseconstraints (dac's) van ziekenhuis verbaal weer.

7.2 Geef de eerste, derde en negende ssr (subset requirement) van ziekenhuis weer in de vorm van een databaseconstraint.

7.3 Geef commentaar/antwoord op de volgende beweringen/vragen.

a. Een verpleegkundige uit Geldrop kan niet behoren tot afdeling 9.
b. Afdeling 9 moet minstens vier verpleegsters hebben.
c. Elke verpleegkundige uit Eindhoven behoort tot afdeling 9.
d. Kan een specialist tot meer dan één afdeling horen?
e. Een patiënt kan voor meer dan één reden worden opgenomen.
f. Worden alleen de 'lopende' opnamen bijgehouden?
g. Een patiënt is iemand, die een opname en/of een behandeling en/of een medicijnverstrekking heeft ondergaan.
h. Het tarief van een behandeling ligt vast door de soort.
i. Het tarief van een behandeling ligt vast door de combinatie van behandelcode en behandelende specialist.
j. Kan worden nagegaan of een specialist actief is geweest?
k. Idem voor een verpleegkundige?
l. De medicijnen met code M12 en M13 moeten altijd worden voorgeschreven voor 5 dagen, 3 maal per dag en 2 stuks per keer.
m. Per medicijnverstrekking liggen aantal dagen, frequentie per dag en aantal stuks per keer vast.
n. Kan een specialist een patiënt zijn?
o. Van elke patiënt is bekend hoeveel malen opname voor 'informaritis severus' heeft plaatsgevonden.

7.4 Ga na voor elk van de vijf onderstaande beweringen a, b, c, d en e, of zij in strijd is met de type-definitie van ziekenhuis. Zo ja, dan moeten *alle* redenen van deze strijdigheid worden gegeven. De variabele zkh is van het type ziekenhuis.

a. #({t(vkanr) | t ∈ zkh(vk)}) = 1.

b. #({t(vknr) | t ∈ zkh(vk(}) = 1.

c. zkh(beh) ⊃ {{(bcd,c1),(bnm,kno6),(bsrt,kno),(btar,05)},
{(bcd,c2),(bnm,kno6),(bsrt,kno),(btar,80)}}.

d. zkh(opn)//$\overline{\{pnr,indat,uitdat\}}$ =
{{(pnr,15),(indat,791207),(uitdat,800407)},
{(pnr,15),(indat,791223),(uitdat,800307)}}.

e. $\{t(pnr) \mid t \in zkh(opn)\} \supset \{15,20,30\} \wedge$

$\{t(pnr) \mid t \in zkh(p)\} = \{6,7,8,15,30,43\}$.

7.5 Bij deze opgave wordt bij elk van de drie onderdelen a, b en c telkens opnieuw uitgegaan van de ziekenhuis-database, zoals gedefinieerd in dit hoofdstuk.
Ga voor elk van de onderdelen na, welke veranderingen in de database-definitie nodig zijn om (met behoud van normaliteit) te voldoen aan het gestelde.

a. Eenzelfde behandeling kan op eenzelfde datum op eenzelfde patiënt niet meer dan eenmaal plaatsvinden.
b. Voor behandelingssoort bsrt = ACDC geldt een vast tarief btar = 40.
c. Voor elke behandelsoort geldt een eigen vast tarief.

8 GEBRUIK EN ONDERHOUD VAN EEN DATABASE; VRAAGTALEN

8.1 INLEIDING

In voorgaande hoofdstukken hebben wij gezien hoe op een 'nette' genormaliseerde wijze de structuur van de gegevens kan worden vastgelegd. Om vervolgens ook werkelijk een (fysiek opgeslagen) database ter beschikking te hebben, zal
- de opslag- en fysieke structuur moeten worden vastgelegd, en
- de database moeten worden 'geladen'.

Met het *laden* van een database wordt bedoeld: het opslaan van de tupels van alle actuele objectoccurrences in het daarvoor aangewezen (achtergrond)geheugen.

Het is eigenlijk niet de bedoeling dat in dit boek de opslag- en de fysieke structuur ter sprake komen en zeker niet het laden van een database. Hiermee wil niet gezegd zijn dat deze onderwerpen niet belangrijk zouden zijn. Integendeel, een goed inzicht in deze zaken is noodzakelijk om met name een goede 'performance' tijdens gebruik en onderhoud van een database mogelijk te maken. Genoemde beperking wordt voor dit boek aangehouden, omdat gebleken is dat het vastleggen van de logische structuur van de gegevens en de manipulatie met de gegevens op hetzelfde logische niveau een duidelijk afgerond onderwerp is van een zodanige omvang en moeilijkheidsgraad, dat aparte behandeling ervan in een boek ten volle gerechtvaardigd is. Overigens zullen wij er in deel III (netwerkmodel) toch niet aan kunnen ontkomen om met name de opslagstructuur nog vrij uitgebreid aan bod te laten komen. Ter plaatse zal duidelijk worden gemaakt waarom dit nodig is.

In dit hoofdstuk zullen wij ons dus beperken tot manipulatie van de gegevens op logisch niveau, dus op het niveau zoals wij ze kennen via de definitie van de logische structuur.

Bij de manipulatie onderscheiden wij, zoals reeds in § 1.3 werd opgemerkt
- *onderhoud* (Engels: maintenance), dat wil zeggen invoegen van nieuwe tupels, veranderen of verwijderen van bestaande tupels;

- *gebruik* (Engels: retrieval) van gegevens.

Gebruik van gegevens is uiteraard de uiteindelijke doelstelling. Maar om steeds de juiste gegevens ter beschikking te hebben, zal over het algemeen veelvuldig onderhoud nodig zijn. Men bedenke namelijk dat de informatiebehoefte dikwijls bij voorkeur betrekking heeft op (in de loop der tijd) veranderende gegevens. Zinvol gebruik dient dan ook vooraf te worden gegaan door adequaat onderhoud.

Overigens menen wij er in dit boek op didactische gronden beter aan te doen eerst het gebruik en dan het onderhoud te behandelen. Alvorens nu met het gebruik (retrieval) te beginnen, eerst nog enkele opmerkingen over zogeheten *vraagtalen*.

Elke zichzelf respecterende leverancier van een DBMS zorgt er heden ten dage voor dat hij een *vraagtaal* (Engels: *query-language*) ter beschikking heeft (SQL, IQL, Query by Example, enz., enz.). De bedoeling van zo'n vraagtaal is dat de gebruiker op een 'user-friendly' manier gebruik kan maken van de database. De benaming 'vraagtaal' is overigens nogal misleidend, want met de geboden faciliteiten kan meestal ook onderhoud worden gepleegd op de database.

Experimenten met beschikbare vraagtalen hebben duidelijk aangetoond dat deze talen toch nog alle van zodanige gebruiksvreemde opzet zijn, dat er bij een iets meer dan elementaire vraag gemakkelijk fouten in kunnen sluipen. Eigenlijk zijn zulke fouten niet te wijten aan de vraagtaal in kwestie. Een vraagtaal is in eerste instantie een neutraal instrument, dat zowel goed als slecht kan worden gebruikt. Vergelijk goede en slechte programma's in overigens respectabele programmeertalen. De eigenlijke oorzaak van genoemde fouten is veeleer te zoeken in een gebrek aan verzamelingstheoretisch inzicht in de gestelde vragen.

Immers, waar gaat het eigenlijk om bij de beantwoording van een vraag. Er wordt dan een verzameling gegevens gevraagd, die is af te leiden uit de verzameling gegevens, die in de database ligt opgeslagen. Onze ervaring is dan ook dat gebruik van vraagtalen veel betrouwbaarder wordt, als men de (ge)vraag(de verzameling) formeel in een verzamelingsexpressie weet weer te geven. Wij zullen daarom vragen (eerst) weergeven in de vorm van verzamelingsexpressies, waarbij dan enkele primitieve 'taalelementen' zullen worden gebruikt. Men bedenke echter, dat er dan geen sprake is van een echte, geïmplementeerde taal. Wat betreft transformatie van de gevonden verzamelingsexpressies naar een weergave in een geïmplementeerde taal, zullen wij aan het einde van dit hoofdstuk voorbeelden geven van transformatie in de vraagtaal SQL en in deel III enkele voorbeelden van transformatie in de DML van het netwerkmodel.

8.2 GEBRUIK (RETRIEVAL)

Wat betreft te gebruiken taalconstructies zullen we ons tot de volgende drie kunnen beperken:

1. <u>get</u> (SE)	SE = set expression (verzamelingsexpressie)
2. <u>def</u> deflist	deflist = lijst van (nieuwe) verzamelingen en/of variabelen (zie voorbeeld 8.7 en 8.8)
3. f*(t) <u>ext</u> db (ob)	betekenis van en nadere toelichting op deze expressie bij voorbeeld 8.10 en 8.11.

Een en ander moge verder duidelijk worden uit de behandeling van onderstaande voorbeelden, die alle betrekking hebben op het in hoofdstuk 7 gedefinieerde databasetype. De variabele zkh is van dit type.

Voorbeeld 8.1

Vraag

Geef naam, adres en woonplaats van de specialisten, die tot afdeling 9 behoren en over meer dan twee bedden kunnen beschikken.

Antwoord

$$\underline{\text{get}}\ (\{t/\overline{\{snm,sadr,swpl\}}\ |\ t \in zkh(sp) \wedge t(spanr) = 9 \wedge t(ab) > 2\}).$$ □

Voor het volgende voorbeeld moeten tabellen van verschillende tabeltypen worden geraadpleegd.

Voorbeeld 8.2

Vraag

Geef naam, adres en woonplaats van de specialisten, die de behandeling met code KNO6 hebben verricht.

Antwoord

$$\underline{\text{get}}\ (\{t/\overline{\{snm,sadr,swpl\}}\ |\ t \in zkh(sp) \wedge \exists u \in zkh(pb)[u(snr) = t(snr) \wedge u(bcd) = KNO6]\}).$$ □

Tussen de vragen van de volgende twee voorbeelden dient goed het verschil te worden gezien. De 'of'-voorwaarde van voorbeeld 8.3 (systologitis òf infologitis) vraagt om een andere aanpak dan de 'en'-voorwaarde van voorbeeld 8.4 (systologitis èn infologitis).

Voorbeeld 8.3

Vraag

Geef naam, adres en woonplaats van de patiënten die in 1980 werden opgenomen voor systologitis of infologitis.

Antwoord 1

$$\underline{\text{get}}\ (\{t\,/\overline{\{pnm, padr, pwpl\}} \mid t \in zkh(p) \wedge$$
$$\exists u \in zkh(opn)\,[\,u(pnr) = t(pnr) \wedge 800101 \leqq u(indat) \leqq 801231 \wedge$$
$$u(opnr) \in \{systologitis, infologitis\}\,]\,\}).$$

Uiteraard is het nu volgende antwoord ook goed, maar in geval er (veel) meer dan twee opnameredenen in het geding zijn, is een antwoord analoog aan antwoord 1 natuurlijk eenvoudiger

Antwoord 2

$$\underline{\text{get}}\ (\{t\,/\overline{\{pnm, padr, pwpl\}} \mid p \in zkh(p) \wedge$$
$$\exists u \in zkh(opn)\,[\,u(pnr) = t(pnr) \wedge 800101 \leqq u(indat) \leqq 801231 \wedge$$
$$(u(opnr) = systologitis \vee$$
$$u(opnr) = infologitis)\,]\,\}).$$

□

Voorbeeld 8.4

Vraag

Geef naam, adres en woonplaats van de patiënten die in 1980 werden opgenomen voor systologitis èn voor infologitis.

Antwoord 1

$$\underline{\text{get}}\ (\{t\,/\overline{\{pnm, padr, pwpl\}} \mid t \in zkh(p) \wedge$$
$$\{systologitis, infologitis\} \subset$$
$$\{u(opnr) \mid u \in zkh(opn) \wedge u(pnr) =$$
$$t(pnr) \wedge 800101 \leqq u(indat) \leqq 801231\}\}).$$

Onderstaand antwoord 2 is ook goed, maar in geval er (veel) meer dan twee opnameredenen in het geding zijn, geldt ook hier weer dat een antwoord analoog aan antwoord 1 veel eenvoudiger is.

Antwoord 2

$$\underline{\text{get}}\ (\{t\,/\overline{\{pnm, padr, pwpl\}} \mid t \in zkh(p) \wedge$$
$$\exists u, v \in zkh(opn)\,[\,u(pnr) = v(pnr) = t(pnr) \wedge$$
$$800101 \leqq u(indat) \leqq 801231 \wedge 800101 \leqq u(indat) \leqq 801231 \wedge$$
$$u(opnr) = systologitis \wedge v(opnr) = infologitis\,]\,\}).$$

□

In de voorwaarden, waaraan tupels van een tabel moeten voldoen, kunnen optreden:

- het aantal elementen van een verzameling;
- de som, het gemiddelde, het maximum, het minimum van een verzameling waarden.

Hiervoor zullen wij gebruik maken van de volgende functies:

\# (reeds bekend) voor het aantal elementen van een verzameling

sum	voor	som	van een verzameling waarden
max		maximum	
min		minimum	
average		gemiddelde	

Als een bepaalde waarde meer dan eens in dezelfde tabel kan optreden (hetgeen voor attributen die geen sleutel zijn altijd mogelijk is), zal men voor het bepalen van som en gemiddelde gebruik moeten maken van de functies: sum2c en average2c. Voor de toelichting hierop zij verwezen naar constraint C4 in voorbeeld 6.3.

Voorbeeld 8.5

Vraag

Geef naam, adres en woonplaats van elke specialist, die in 1980 meer dan 10 maal de behandeling KNO7 heeft verricht en die voor deze behandeling in 1980 gemiddeld meer dan een half uur nodig had.

Antwoord

$$\underline{\text{get}}\ (\{t\,/\,\overline{\{snm,sadr,swpl\}}\ \mid\ t \in zkh(sp)\ \wedge$$
$$\#(\{u\ \mid\ u \in zkh(pb) \wedge u(snr) = t(snr)\ \wedge$$
$$800101 \leqq u(dat) \leqq 801231 \wedge u(bcd) = KNO7\}) > 10\ \wedge$$
$$average2c(\{(u,u(duur))\ \mid\ u \in zkh(pb)\ \wedge$$
$$800101 \leqq u(dat) \leqq 801231 \wedge u(bcd) = KNO7\ \wedge$$
$$u(snr) = t(snr)\}) > 30\}). \qquad \square$$

Bovengenoemde functies kunnen niet alleen onderdeel vormen van een voorwaarde, maar ook van het gevraagde zelf, zoals in het volgende voorbeeld.

Voorbeeld 8.6

Vraag

Geef van elke specialist, die in 1980 meer dan 10 maal de behandeling KNO7 heeft verricht: naam, adres, woonplaats, het aantal malen dat hij deze behandeling heeft verricht in 1980 en de maximale en gemiddelde duur van dit aantal.

Antwoord

```
get ({(t/{snm,sadr,swpl},
        #({u | u ∈ zkh(pb) ∧ u(snr) = t(snr) ∧ u(bcd) = KNO7 ∧
                     800101 ≤ u(dat) ≤ 801231}),
        max({u(duur) | u ∈ zkh(pb) ∧ u(snr) = t(snr) ∧
              u(bcd) = KNO7 ∧ 800101 ≤ u(dat) ≤ 801231}),
        average2c({(u,u(duur)) | u ∈ zkh(pb) ∧
                     u(snr) = t(snr) ∧ u(bcd) = KNO7 ∧
                     800101 ≤ u(dat) ≤ 801231})) | t ∈ zkh(sp) ∧
              #({ u | u ∈ zkh(pb) ∧ u(snr) = t(snr) ∧
              u(bcd) = KNO7 ∧ 800101 ≤ u(dat) ≤ 801231}) > 10}).
```

Voor een ander antwoord op deze zelfde vraag, zie voorbeeld 8.8. □

Expressies, zoals in bovenstaand voorbeeld, zijn zo omvangrijk en ingewikkeld, dat 'splitsing' in 'tussenverzamelingen' eigenlijk mogelijk zou moeten zijn. Zo'n splitsing is inderdaad mogelijk via aparte definities in het def-gedeelte. Wij zullen dit duidelijk maken aan de hand van het volgende voorbeeld. Verder wordt dan in voorbeeld 8.8 een alternatieve oplossing gegeven van de vraag in voorbeeld 8.6.

Voorbeeld 8.7

Vraag

Geef naam, adres en woonplaats van de specialisten, die in 1980 minstens eenmaal behandeling B13 hebben uitgevoerd, zodanig dat deze behandeling langer duurde dan enige behandeling B13 in 1979 door welke specialist ook.

In antwoord1 wordt een oplossing gegeven zonder 'splitsing', dus zonder een def-gedeelte met aparte definities. In antwoord2 wordt wel gewerkt met een def-gedeelte.

Antwoord 1

```
get ({t/{snm,sadr,swpl} | t ∈ zkh(sp) ∧
      ∃u ∈ zkh(pb)[u(snr) = t(snr) ∧ 800101 ≤ u(dat) ≤ 801231 ∧
                  u(bcd) = B13 ∧ u(duur) >
                  max({v(duur) | v ∈ zkh(pb) ∧ v(bcd) = B13 ∧
                  790101 ≤ v(dat) ≤ 791231})]}).
```

Antwoord 2

$$
\begin{aligned}
&\underline{\text{def}}\ H := \{t/\{snr,duur\}\ |\ t \in zkh(pb)\ \wedge \\
&\qquad 800101 \leq t(dat) \leq 801231 \wedge t(bcd) = B13\}; \\
&\qquad maxd79 := max(\{t(duur)\ |\ t \in zkh(pb)\ \wedge \\
&\qquad\qquad\qquad t(bcd) = B13 \wedge 790101 \leq t(dat) \leq 791231\}) \\
&\underline{\text{get}}\ (\{t/\{snm,sadr,swpl\}\ |\ t \in zkh(sp)\ \wedge \\
&\qquad \exists u \in H[u(snr) = t(snr) \wedge u(duur) > maxd79]\}). \qquad \square
\end{aligned}
$$

Voorbeeld 8.8

Vraag

Als in voorbeeld 8.6.

Antwoord

$$
\begin{aligned}
&\underline{\text{def}}\ H := \{t/\{bcd,pnr,dat,duur,snr\}\ |\ t \in zkh(pb)\ \wedge \\
&\qquad\qquad 800101 \leq t(dat) \leq 801231\})\} \\
&\underline{\text{get}}\ (\{(t/\{snm,sadr,swpl\}, \#(\{u\ |\ u \in H \wedge u(snr) = t(snr)\}), \\
&\qquad max(\{u(duur)\ |\ u \in H \wedge u(snr) = t(snr)\}), \\
&\qquad average2c(\{(u,u(duur))\ |\ u \in H \wedge u(snr) = t(snr)\}))\ | \\
&\qquad\quad t \in zkh(sp) \wedge \#(\{u\ |\ u \in H \wedge u(snr) = t(snr)\}) > 10\}). \qquad \square
\end{aligned}
$$

In het databasetype van hoofdstuk 7 zijn vele ssr's (subset requirements) gedefinieerd. Deze garanderen, zoals gezegd, dat naast tupels van een bepaald object andere tupels van een ander object beschikbaar zullen zijn.

Voorbeeld 8.9

Vraag

Geef naam, adres en woonplaats van de specialisten, die in december 1980 verantwoordelijk zijn geweest voor een opname van een patiënt uit Eindhoven.

Antwoord

$$
\begin{aligned}
&\underline{\text{get}}\ (\{t/\{snm,sadr,swpl\}\ |\ t \in zkh(sp)\ \wedge \\
&\qquad \exists u \in zkh(opn)[801201 \leq u(indat) \leq 801231\ \wedge \\
&\qquad\qquad\qquad u(snr) = t(snr)\ \wedge \\
&\qquad\qquad \exists v \in zkh(p)[v(pnr) = u(pnr)\ \wedge \\
&\qquad\qquad\qquad\qquad v(pwpl) = eindhoven]]\}). \qquad \square
\end{aligned}
$$

In bovenstaand voorbeeld 'klinkt het een beetje overdreven', dat als conditie wordt gesteld: "$\exists v \in zkh(p)[v(pnr) = u(pnr)\ldots$", terwijl wij vanwege de subset requirement (opn.{pnr},p.{pnr}, id) zeker weten dat er zo'n p-tupel moet zijn. Ofschoon bovenstaand antwoord niet fout is, gaan we toch meer expliciet gebruik maken van de ssr's. We zullen dit eerst demonstreren aan een voorbeeld, en dan een algemene aanpak schetsen.

Stel: pat is de attribtrafo {(pnr,pnr)}, en t is een tupel, waarin pnr als attribuut voorkomt zoals een opname- of een patiëntbehandelingstupel. Wij definiëren nu de zogeheten sterfunctie pat* als volgt:

$$pat^*(t) := \{(pat(pnr), t(pnr))\}, \text{ dus}$$

$$pat^*(t) := \{(pnr, t(pnr))\}.$$

Met pat*(t) ligt nu precies één tupel vast van het type patiënt, daar {pnr} sleutel is van patiënt. We kunnen pat*(t) dus eenduidig 'uitbreiden' tot een patiënttupel en deze uitbreiding wordt nu bedoeld met de uitdrukking:

$$pat^*(t)\ \underline{ext}\ zkh(p).$$

Dus (pat*(t) ext zkh(p)) is het patiënttupel t met t(pnr) als waarde voor de sleutel pnr. Dan is dus ook:

$$(pat^*(t)\ \underline{ext}\ zkh(p))(pwpl) = eindhoven$$

een syntactisch juiste bewering.

Met behulp van het bovenstaande kunnen wij nu een antwoord geven voor de vraag in voorbeeld 8.9, waarbij expliciet gebruik wordt gemaakt van een van de ssr's (met bijbehorende attribtrafo).

Voorbeeld 8.10

Vraag

Als in voorbeeld 8.9.

Antwoord

$$\underline{get}\ (\{t\sqrt{\{snm,sadr,swpl\}} \mid t \in zkh(sp) \wedge \exists u \in zkh(opn)[\,801201 \leq u(indat) \leq 801231 \wedge u(snr) = t(snr) \wedge (pat^*(u)\ \underline{ext}\ zkh(p))(pwpl) = eindhoven\,]\}).$$

□

Na bovenstaand voorbeeld zullen wij nu het gebruik van ssr's door middel van sterfuncties algemeen behandelen voor een databasetype.

Zij gegeven databasetype DT en de variabele DB van dit type. Verder is f een attribtrafo met dom(f) = A1 en rng(f) = A2. De sterfunctie f* wordt nu als volgt gedefinieerd:

- dom(f*) is de verzameling van alle tupels u, waarvan de attributenverzameling A1 bevat èn
- f*(u) = {(f(a),u(a)) | a ∈ A1}.

Als f nu gebruikt wordt in een ssr vanuit A1 van OB1 naar A2 in OB2, zodanig dat A1 een referentiesleutel is ten opzichte van A2, dan bepaalt f*(u) een sleutelwaarde van OB2. Met

f*(u) ext DB(OB2)

wordt dan bedoeld: het door f*(u) uniek bepaalde tupel van DB(OB2).

Van het bovenstaande zullen wij gebruik kunnen maken bij het oplossen van vragen op de ziekenhuis-database. De in hoofdstuk 7 gebruikte attribtrafo's bij de ssr-definities zullen wij als volgt vastleggen:

```
pat   := {(pnr,pnr)}
vk    := {(vknr,vknr)}
spec  := {(snr,snr)}
afd1  := {(vkanr,anr)}
afd2  := {(spanr,anr)}
afd3  := {(vanr,anr)}
vr    := {(vnr,vnr)}
opn   := {(pnr,pnr),(indat,indat)}
beh   := {(bcd,bcd)}
med   := {(mcd,mcd)}.
```

Bij al deze attribtrafo's is A1 een referentiesleutel ten opzichte van A2. Dit is niet toevallig, aangezien ziekenhuis een genormaliseerd databasetype is. Bij een genormaliseerd databasetype zijn eigenlijk alleen die ssr's zinvol, die uniek 'heen wijzen' naar tupels van andere objecten.

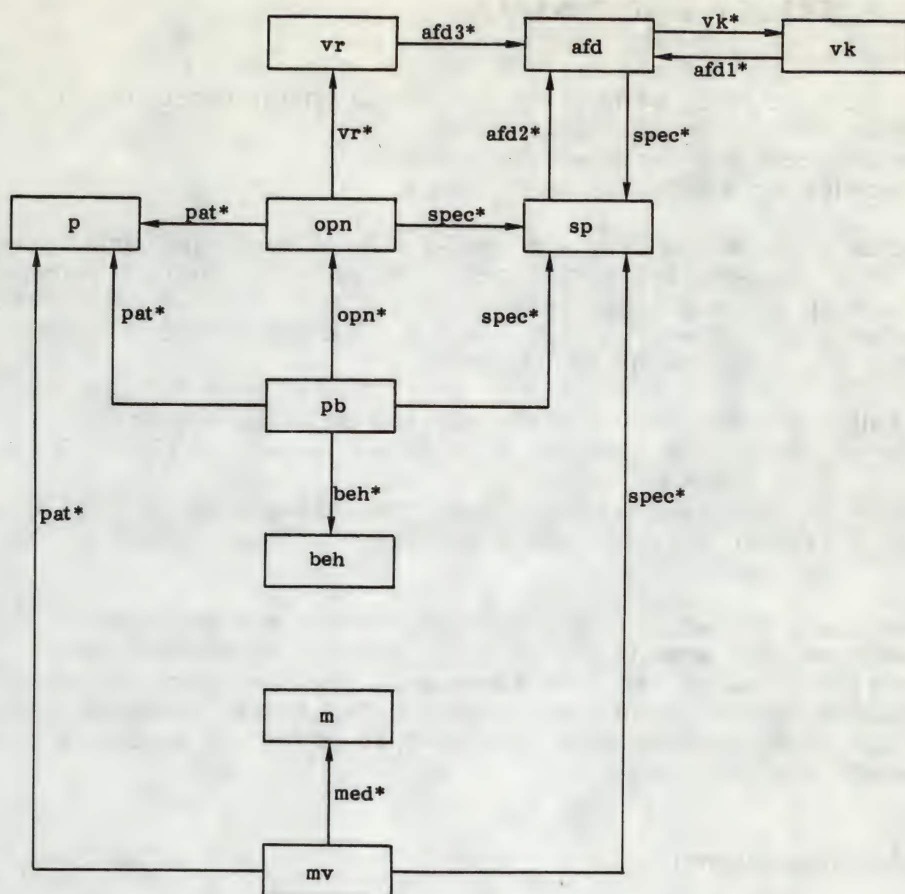
In het diagram op p.103 zijn de bijbehorende sterfuncties (pat*, enz.) weergegeven.

We zullen deze paragraaf nu besluiten met een voorbeeld, waarin 'uitbundig' gebruik kan worden gemaakt van sterfuncties en waarin ook het 'nesten' van sterfuncties voorkomt.

Voorbeeld 8.11

Vraag

Geef van elke behandeling die in mei 1980 werd verricht door een specialist van de afdeling dermatologie: behandelcode en -soort, patiëntnummer en -naam, datum, specialistnummer en -naam.



Figuur 8.1 Diagram van ziekenhuis-database met sterfuncties.

Antwoord

```
get ({{(bcd,t(bcd),
      (bsrt,(beh*(t) ext zkh(beh))(bsrt)),
      (pnr,t(pnr)),
      (pnm,(pat*(t) ext zkh(p))(pnm)),
      (dat,t(dat)),
      (snr,t(snr)),
      (snm,(spec*)(t) ext zkh(sp))(snm))} |

        t ∈ zkh(pb) ∧ 800501 ≦ t(dat) ≦ 800531 ∧
        (afd2*(spec*(t) ext zkh(sp)) ext zkh(afd))(anm) =
                                dermatologie}).
```

□

8.3 ONDERHOUD (MAINTENANCE)

In § 8.1 werd reeds vermeld dat onderhoud (maintenance) omvat
- invoegen van nieuwe tupels en/of
- veranderen van bestaande tupels en/of
- verwijderen van bestaande tupels.

Evenals bij de behandeling van retrieval in de vorige paragraaf zullen wij bij de behandeling van onderhoud gebruik maken van verzamelingsexpressies tezamen met enkele primitieve 'taalelementen'. Ook hier bedenke men weer, dat er met deze taalelementen geen sprake is van een echte, geïmplementeerde taal.

Wij zullen gebruik maken van de volgende taalconstructies:
1. insert(INTT,TT), voor het invoegen van tupels; INTT en TT zijn tabellen.
2. delete(DTT), voor het verwijderen van de tupels in de tabel DTT.
3. modify(MSE), voor het veranderen van tupels met behulp van de M(utatie) SE

Voorafgaand aan elk van deze drie constructies kan eventueel gebruik worden gemaakt van de reeds bekende def constructie.
Wij zullen nu elk van de onderhoudsopdrachten apart bespreken. Wij zullen daarbij steeds veronderstellen dat zkh een variabele is van het databasetype ziekenhuis (hoofdstuk 7) en dat zkh reeds een bepaalde waarde heeft.

8.3.1 Invoegen (insert)

Wij beginnen met een introductie via een voorbeeld.

Voorbeeld 8.12

Opdracht

Voeg de volgende twee opnametupels, als weergegeven in onderstaande tabel, in.

pnr	opnr	vnr	indat	uitdat	snr
15	datalogitis	26	820614	991231	36
27	datalogitis	28	820614	991231	36

Oplossing 1

```
insert({{(pnr,15),(opnr,datalogitis),(vnr,26),
          (indat,820614),(uitdat,991231),(snr,36)},
        {(pnr,27),(opnr,datalogitis),(vnr,28),
          (indat,820614),(uitdat,991231),(snr,36)}}, zkh(opn))
```

Oplossing 2

def INOPN := {{(pnr,15),(opnr,datalogitis),(vnr,26),
(indat,820614),(uitdat,991231),(snr,36)},
{(pnr,27),(opnr,datalogitis),(vnr,28),
(indat,820614),(uitdat,991231),(snr,36}}
insert(INOPN, zkh(opn))

Bovenstaande invoegopdracht is gelijk aan de opdracht:

zkh(opn) := zkh(opn) ∪ INOPN □

Hiervoor werd reeds de algemene vorm van de insert-opdracht gegeven, namelijk insert(INTT,TT). Hierbij zijn TT en INTT tabelvariabelen met hetzelfde domein.

De insert-opdracht is gelijk aan de opdracht:

TT := TT ∪ INTT.

Of een insert-opdracht uitvoerbaar is, hangt af van het feit of TT ∪ INTT een tabel is, die niet strijdig is met het gegeven databasetype.

8.3.2 Verwijderen (delete)

Ook hier beginnen wij weer met een voorbeeld.

Voorbeeld 8.13

Opdracht

Verwijder de patiëntbehandelingstupels, die betrekking hebben op de patiënt met patiëntnummer 82 en op een behandelingsdatum in 1980.

Oplossing 1

delete({t | t ∈ zkh(pb) ∧ t(pnr) = 82 ∧
800101 ≤ t(dat) ≤ 801231})

Oplossing 2 (met deflist)

def DTT := {t | t ∈ zkh(pb) ∧ t(pnr) = 82 ∧
800101 ≤ t(dat) ≤ 801231}

delete(DTT).

Het resultaat van bovenstaande verwijder-opdracht is gelijk aan het resultaat van de opdracht:

zkh(pb) := zkh(pb) ∖ DTT □

In het algemeen heeft de verwijderopdracht de vorm:

$\underline{\text{delete}}(\{t \mid t \in TT \wedge P(t)\})$,   waarbij

TT een tabelvariabele is, en
P een predikaat over de tupels van TT.

De delete-opdracht heeft hetzelfde resultaat als de opdracht:

$TT := TT \setminus \{t \mid t \in TT \wedge P(t)\}$

Over de uitvoerbaarheid van een delete-opdracht moet, evenals bij de insert-opdracht, worden opgemerkt, dat deze afhankelijk is van het feit of de nieuwe waarde van TT een tabel is, die niet strijdig is met het gegeven databasetype.

8.3.3 Veranderen (modify)

Eerst weer een voorbeeld.

Voorbeeld 8.14

Opdracht

Verander in het opnametupel met patiëntnummer 15 en indat 820614 de waarde van uitdat in 820625.

Oplossing 1

$$\underline{\text{modify}}(\{(t,\{(uitdat,820625)\}) \mid t \in zkh(opn) \wedge t(pnr) = 15 \wedge t(indat) = 820614\})$$

Oplossing 2

$$\underline{\text{def}}\ MTT := \{t \mid t \in zkh(opn) \wedge t(pnr) = 15 \wedge t(indat) = 820614\}$$

$$\underline{\text{modify}}(\{(t,\{(uitdat,820625)\}) \mid t \in MTT\}).$$

Het resultaat van bovenstaande modify-opdracht is gelijk aan het resultaat van de volgende opdracht:

$$zkh(opn) := (zkh(opn) \setminus MTT) \cup \{t/\overline{dom(t) \setminus \{uitdat\}} \cup \{(uitdat,820625)\} \mid t \in MTT\}$$ □

Bovenstaand voorbeeld is op twee manieren uitbreidbaar:
- de verandering van de waarde van een attribuut is afhankelijk

van de bestaande waarde, bijvoorbeeld: tarief wordt 1,5 maal zo groot;
- meer dan een attribuut moet een verandering ondergaan, bijvoorbeeld adres èn woonplaats van een patiënt.

Van beide uitbreidingen volgt nu een voorbeeld.

Voorbeeld 8.15

Opdracht

Verander adres, woonplaats van specialist 12 in dalweg 18, geldrop en verander zijn afdelingsnummer in 7.

Oplossing

modify({(t,{(sadr,dalweg 18),(swpl,geldrop),(spanr,7)} |
t ∈ zkh(sp) ∧ t(snr) = 12}) □

Voorbeeld 8.16

Opdracht

Verminder van elke specialist met een aantal bedden > 0, dit aantal met 1.

Oplossing

modify({(t,{(ab,t(ab) - 1)}) | t ∈ zkh(sp) ∧ t(ab) > 0})

□

Na het bovenstaande kunnen wij nu overgaan op de algemene vorm van een modify-opdracht. Deze is:

modify({(t,mf(t) | t ∈ TT ∧ P(t)}),

waarbij mf(t) een zogeheten *modificatiefunctie op t* is. Het domein van deze functie is de verzameling MA van alle te veranderen attributen en voor elk element ma van MA geldt dat mf(t)(ma) een uitdrukking is in t. (Deze uitdrukking kan ook een constante zijn, zoals in de voorbeelden 8.14 en 8.15.)

Uit het voorgaande is eenvoudig onderstaand tabelletje af te leiden.

voorbeeld	modificatiefunctie
8.14	{(uitdat,820625)}
8.15	{(sadr,dalweg 18),(swpl,geldrop),(spanr,7)}
8.16	{(ab,t(ab)-1)}

Het resultaat van een modify-opdracht is gelijk aan het resultaat van de volgende opdracht:

$$TT := (TT \setminus MTT) \cup \{t/\overline{dom(t) \setminus MA} \cup \{(ma, mf(t)(ma)) \mid ma \in MA\} \mid t \in MTT\},$$

waarbij MTT = $\{t \mid t \in TT \wedge P(t)\}$
mf(t): de modificatiefunctie op t
MA: het domein van mf(t).

Over de uitvoerbaarheid van een modify-opdracht geldt dezelfde opmerking als over de uitvoerbaarheid van een insert- en van een delete-opdracht.

8.4 TRANSFORMATIE NAAR EEN VRAAGTAAL

Zoals reeds in de eerste paragraaf van dit hoofdstuk werd vermeld, zullen wij nu enkele vragen en onderhoudsopdrachten beantwoorden en oplossen met behulp van de vraagtaal SQL. Men beschouwe dit als een voorbeeld van transformatie naar een bestaande vraagtaal. Uiteraard kan men een soortgelijke transformatie (trachten te) ondernemen met elke andere vraagtaal. De keus moet in dit opzicht verder aan de lezer zelf worden overgelaten, aangezien ter plaatse beschikbare faciliteiten een overheersende rol kunnen spelen.

Voor de behandeling van de taal SQL zij verwezen naar DATE (hst. 7 en 9) en/of naar een SQL-user manual.

Wij zullen nu (trachten) verschillende vragen van § 8.2 in SQL weer (te) geven, en deze weergave waar nodig van commentaar voorzien. Voor de eenvoud van vergelijking zullen wij in deze paragraaf voor overeenkomstige voorbeelden dezelfde nummering aanhouden plus de toevoeging S. Dus voorbeeld 8.1S is het SQL-equivalent van voorbeeld 8.1.

Voorbeeld 8.1S

Vraag

Geef naam, adres en woonplaats van de specialisten, die tot afdeling 9 behoren en over meer dan twee bedden kunnen beschikken.

Antwoord

```
SELECT  SNM,SADR,SWPL
FROM    SP
WHERE   SPANR = 9 AND AB > 2
```

Commentaar
Transformatie naar SQL 'recht-toe-recht-aan'; alleen wordt in SQL nooit de databasevariabele genoemd. Dus niet FROM ZKH.SP, maar alleen FROM SP.

Voorbeeld 8.2S

Vraag

Geef naam, adres en woonplaats van de specialisten, die de behandeling met code KNO6 hebben verricht.

Antwoord

```
SELECT  SNM,SADR,SWPL
FROM    SP
WHERE  ( SNR IN
       | SELECT  SNR
     A {  FROM    PB
       ( WHERE   BCD = "KNO6"
```

Commentaar
Het stuk A is het equivalent van "$\exists u \in zkh(pb)[u(snr) = t(snr) \wedge u(bcd) = KNO6]$", hetgeen overigens zelf equivalent is met "$t(snr) \in \{u(snr) \mid u \in zkh(pb) \wedge u(bcd) = KNO6$", en dit laatste is eigenlijk letterlijk de weergave sub A. In SQL kan verder de existentiële quantor worden gerepresenteerd met behulp van de EXISTS-clausule. Het antwoord op de vraag van dit voorbeeld wordt dan

```
SELECT  SNM,SADR,SWPL
FROM    SP
WHERE   EXISTS
        (SELECT  *
        FROM     PB
        WHERE    SNR = SP.SNR AND BCD = "KNO6")
```
□

Voorbeeld 8.3S

Vraag

Geef naam, adres en woonplaats van de patiënten die in 1980 werden opgenomen voor systologitis of infologitis.

Antwoord

```
SELECT  PNM,PADR,PWPL
FROM    P
WHERE   PNR IN
        SELECT  PNR
        FROM    OPN
        WHERE   OPNR IN < "SYSTOLOGITIS","INFOLOGITIS" >
```
□

Voorbeeld 8.4S

Vraag

Geef naam, adres en woonplaats van de patiënten die in 1980 werden opgenomen voor systologitis èn voor infologitis.

Antwoord

```
SELECT  PNM,PADR,PWPL
FROM    P
WHERE   < "SYSTOLOGITIS","INFOLOGITIS" > IN
       ⎧SELECT  OPNR
      B⎨FROM    OPN
       ⎪WHERE   PNR = P.PNR AND
       ⎩        800101 ≦ INDAT ≦ 801231.
```

Commentaar
Ten overvloede zij opgemerkt, dat B de verzameling is van alle opnameredenen in 1980 van de 'aan de beurt zijnde' patiënt. □

Nu komen wij aan het gebruik van de functies #, sum, max, min, average. De (min of meer) equivalenten hiervan in SQL zijn de zogeheten built-in functions COUNT, SUM, MAX, MIN, AVG.

Voorbeeld 8.5S

Vraag

Geef naam, adres en woonplaats van elke specialist, die in 1980 meer dan 10 maal de behandeling KNO7 heeft verricht en die voor deze behandeling in 1980 gemiddeld meer dan een half uur nodig had.

Antwoord

```
SELECT  SNM,SADR,SWPL
FROM    SP
WHERE   (SELECT COUNT(*)
        FROM    PB
        WHERE   SNR = SP.SNR AND
                800101 ≦ DAT ≦ 801231 AND
                BCD = "KNO7") > 10
        AND
        (SELECT AVG(DUUR)
        FROM    PB
        WHERE   SNR = SP.SNR AND
                800101 ≦ DAT ≦ 801231 AND
                BCD = "KNO7") > 30
```

Commentaar
- COUNT(*) betekent: tel aantal tupels van de tabel onder FROM.
- In SQL betekent AVG(DUUR) het gemiddelde over alle voorkomende waarden van dit attribuut, waarbij meermalen voorkomende

waarden ook meermalen meetellen. Wil men meermalen voorkomende waarden maar éénmaal meetellen, dan moet AVG(UNIQUE DUUR) worden gebruikt. □

Voor een SQL-equivalent van voorbeeld 8.6 zie verderop voorbeeld 8.8S.

Voorbeeld 8.7S

Vraag

Geef naam, adres en woonplaats van de specialisten, die in 1980 minstens éénmaal een behandeling B13 hebben uitgevoerd, zodanig dat deze behandeling langer duurde dan enige behandeling B13 in 1979 door welke specialist ook.

Waar in het antwoord van voorbeeld 8.7 sprake was van splitsing met behulp van de def-clausule, kan dit in SQL ook (tenminste wat betreft tabellen) en wel met behulp van de zogeheten DEFINE VIEW-clausule.

Antwoord

```
DEFINE  VIEW H AS
        SELECT  UNIQUE SNR,DUUR
        FROM    PB
        WHERE   800101 ≦ DAT ≦ 801231 AND
                BCD = "B13".

SELECT  SNM,SADR,SWPL
FROM    SP
WHERE   SNR IN
        SELECT  SNR
        FROM    H
        WHERE   DUUR >
                SELECT  MAX(DUUR)
                FROM PB
                WHERE 790101 ≦ DAT ≦  791231 AND
                      BCD = "B13".
```
□

Voorbeeld 8.8S

Vraag

Geef van elke specialist, die in 1980 meer dan 10 maal de behandeling KNO7 heeft verricht: naam, adres, woonplaats, het aantal malen dat hij deze behandeling heeft verricht in 1980 en de maximale en gemiddelde duur van dit aantal.

Antwoord

```
DEFINE  VIEW SPBEH(SNR,AANTAL,MAXDUUR,GEMDUUR) AS
        SELECT  SNR,COUNT(*),MAX(DUUR),AVG(DUUR)
        FROM    PB
        WHERE   800101 ≤ DAT ≤ 801231 AND BCD = "KNO7"
        GROUP BY SNR
        HAVING COUNT(*) > 10

SELECT  SNM,SADR,SWPL,AANTAL,MAXDUUR,GEMDUUR
FROM    SP,SPBEH
WHERE   S.SNR = SPBEH.SNR
```

In SQL kan het antwoord op deze vraag niet anders dan in 'opgesplitste' vorm, dus met een DEFINE-gedeelte, worden gegeven. Een antwoord, analoog aan dat van voorbeeld 8.6, is dus niet mogelijk in SQL. Dit is te wijten aan het feit, dat de parameters (variabelen) van de built-in functions niet bij de functie-aanroep zelf vermeld staan, maar 'daarbuiten her en der verspreid liggen' en daardoor moeilijk of in het geheel niet als zodanig herkenbaar zijn. Deze 'eigenschap' van de built-in functions kan tot vrij ingewikkelde antwoorden leiden. Stel bijvoorbeeld dat bovenstaande vraag vervangen wordt door de volgende:

Geef van elke specialist: naam, adres, woonplaats en het aantal malen, dat hij in 1980 de behandeling KNO7 heeft verricht.

Beantwoording in het verzamelingsmodel kan eenvoudig geschieden door uit het antwoord van voorbeeld 8.6 enkele regels weg te laten, zodat men krijgt:

$\underline{\text{get}}\ (\{(t\ /\overline{\{snm,sadr,swpl\},}$
$\quad \#(\{u \mid u \in zkh(pb) \wedge u(snr) = t(snr) \wedge$
$\qquad 800101 \leq u(dat) \leq 801231 \wedge$
$\qquad\quad u(bcd) = KNO7\}) \mid t \in zkh(sp)\}).$

Een antwoord in SQL, op analoge wijze met weglating van 'overeenkomstige' gedeelten, zou leiden tot:

```
DEFINE  VIEW SPBEH(SNR,AANTAL) AS
        SELECT  SNR,COUNT(*)
        FROM    PB
        WHERE   800101 ≤ DAT ≤ 801231 AND BCD = "KNO7"
        GROUP BY SNR

SELECT  SNM,SADR,SWPL,AANTAL
FROM    SP,SPBEH
WHERE   S.SNR = SPBEH.SNR.
```

Met dit antwoord zouden dan echter de specialisten, die nog geen enkele behandeling hebben verricht, onvermeld blijven.

Een juist antwoord in SQL kan worden verkregen door een nieuwe tabel (object) of (iets eenvoudiger) een nieuw veld (attribuut) te 'creëren'. Wij geven hier het antwoord met behulp van een nieuw veld.

```
EXPAND TABLE SP
       ADD AANTAL INTEGER

UPDATE SP
SET    AANTAL = 0

UPDATE SP
SET    AANTAL =
       (SELECT COUNT(*)
       FROM   PB
       WHERE  SNR = SP.SNR AND 800101 ≦ DAT ≦ 801231
       AND    BCD = "KNO7")
WHERE  SNR IN
       (SELECT SNR
       FROM   PB)

SELECT SNM,SADR,SWPL,AANTAL
from   SP.
```

Met de voorbeelden 8.9, 8.10 en 8.11 bevinden wij ons op het gebied van de ssr's en dus in het geval van een genormaliseerd databasetype, op het gebied van de externe sleutels. Dit begrip is (bij mijn weten) onbekend in SQL. Om te demonstreren hoe men hiermee in SQL 'terecht' komt, moge een behandeling van voorbeeld 8.11 volstaan.

Voorbeeld 8.11S

Vraag

Geef van elke behandeling die in mei 1980 werd verricht door een specialist van de afdeling dermatologie: behandelcode en -soort, patiëntnummer en -naam, datum, specialistnummer en -naam.

Antwoord

```
SELECT BCD,BSRT,P.PNR,PNM,DAT,SP.SNR,SNM
FROM   PB,P,SP,BEH
WHERE  800501 ≦ DAT ≦ 800531 AND
       PB.PNR = P.PNR AND
       PB.SNR = SP.SNR AND PB.BCD = BEH.BCD AND
       SP.SPANR IN
              SELECT ANR
              FROM   AFD
              WHERE  ANM = "DERMATOLOGIE".
```
□

Wij zullen nu enkele onderhoudsopdrachten van § 8.3 in SQL weergeven.

Voorbeeld 8.12S

Opdracht

Voeg de volgende twee opnametupels, als weergegeven in onderstaande tabel, in.

pnr	opnr	vnr	indat	uitdat	snr
15	datalogitis	26	820614	991231	36
27	datalogitis	28	820614	991231	36

Oplossing

```
INSERT  INTO OPN(PNR,OPNR,VNR,INDAT,UITDAT,SNR):
        < 15,"DATALOGITIS",26,820614,991231,36 >

INSERT  INTO OPN(PNR,OPNR,VNR,INDAT,UITDAT,SNR):
        < 27,"DATALOGITIS",28,820614,991231,36 >.
```

Commentaar
In SQL is invoeging alleen mogelijk op tupelniveau. Er zijn dan ook twee INSERT-opdrachten nodig.
(Pseudo-invoeging, door 'overname' uit andere tabellen, kan wel op tabelniveau.) □

Voorbeeld 8.13S

Opdracht

Verwijder de patiëntbehandelingstupels die betrekking hebben op de patiënt met patiëntnummer 82 en op een behandelingsdatum in 1980.

Oplossing

```
DELETE  PB
WHERE   PNR = 82 AND 800101 ≤ DAT ≤ 801231.
```
□

Van de veranderingsopdrachten zullen wij alleen voorbeeld 8.15 in SQL behandelen.

Voorbeeld 8.15S

Opdracht

Verander adres, woonplaats van specialist 12 in dalweg 18, geldrop en verander zijn afdelingsnummer in 7.

Oplossing

```
UPDATE SP
SET    SADR = "DALWEG 18",
       SWPL = "GELDROP",
       SPANR = 7
WHERE  SNR = 12.
```
□

Samenvattend kan het volgende worden opgemerkt over de transformatie naar SQL:
- mits geen built-in functions nodig zijn, geeft de transformatie weinig problemen;
- als built-in functions wel nodig zijn, kan het erg lastig worden de juiste transformatie tot stand te brengen;
- in eerste instantie is niet de taal het belangrijkste, maar een goede verzamelingsanalyse van het gestelde probleem.

OPGAVEN

Alle opgaven hebben betrekking op een ziekenhuis-database volgens de definitie van hoofdstuk 7.
Voor de opgaven 8.1-8.30: antwoord volgens het verzamelingsmodel (en naar believen in één of meer vraagtalen).

8.1 Geef alle gegevens van de patiënten die in Eindhoven wonen.

8.2 Geef nummer en naam van de specialisten, die tussen 1 november 1978 en 1 februari 1979 verantwoordelijk zijn geweest voor een opname.

8.3 Geef alle gegevens van de patiënten die tussen 1 november 1978 en 1 februari 1979 werden opgenomen in afdeling 12.

8.4 Geef van elke afdeling: nummer, naam, aantal verpleegkundigen en aantal specialisten.

8.5 Geef nummer en naam van elke afdeling waarvoor geldt dat het totale aantal voor specialisten beschikbare bedden minder dan 20% van het totale aantal beschikbare bedden bedraagt.

8.6 Geef de gegevens van de specialisten, die in januari 1979 verantwoordelijk zijn geweest voor de opnamen in afdeling 5.

8.7 Geef nummer en naam van de afdelingen, waarvan hoofdspecialist en hoofdverpleegkundige niet in dezelfde plaats wonen.

8.8 Geef nummer en naam van de patiënten die zowel medicijn M12A als medicijn M12C hebben gebruikt.

8.9 Geef van elke afdeling met meer dan twee verpleegruimten: nummer, naam en totaal aantal beschikbare bedden.

8.10 Geef nummer en naam van de specialisten, die aan patiënten die werden opgenomen voor 'helioritis' een medicijn van de soort MSX18 hebben voorgeschreven.

8.11 Geef nummer en naam van de patiënten die de behandelingen B26, B13, B26, B30 in deze volgorde hebben ondergaan.

8.12 Geef nummer en naam van de specialisten, die minstens alle behandelingen hebben verricht, die ook door specialist 12 zijn verricht.

8.13 Geef nummer en naam van elke specialist, die minstens twee verschillende soorten behandelingen heeft verricht (let wel: soort is niet gelijk aan code).

8.14 Geef nummer en naam van elke afdeling, waaraan minstens twee specialisten verbonden zijn die in juni 1979 de behandeling B09 hebben verricht.

8.15 Geef van elke specialist, die behandelingen van soort BS2X heeft verricht: nummer, naam, aantal malen dat hij deze soort behandeling heeft verricht, maximale duur, gemiddelde duur.

8.16 Geef nummer en naam van elke afdeling, waarin in de week van 11 tot en met 17 februari 1979 geen patiënten uit Geldrop werden opgenomen.

8.17 Geef nummer en naam van elke patiënt, die in 1978 niet in twee of meer verschillende afdelingen heeft gelegen.

8.18 Geef alle gegevens van de patiënten, die werden behandeld door (minstens) een specialist, die een behandeling verrichtte op (minstens) een patiënt, die werd behandeld door (minstens) een specialist van afdeling 9.

8.19 Geef nummer en naam van de specialisten van afdeling 15, die in maart 1979 een patiënt hebben behandeld in een behandelruimte, waarin in februari 1979 een patiënt een behandeling heeft ondergaan van de soort BS15A7.

8.20 Geef nummer en naam van de patiënten, die werden opgenomen in afdeling 17 en (tijdens dezelfde opname), na minstens tweemaal verstrekking van minstens vijf stuks van het medicijn MX13 een behandeling van de soort BS07 hebben ondergaan.

8.21 Geef nummer en naam van de patiënten die in januari 1979 werden opgenomen in een afdeling waarin in december 1978 geen patiënten werden opgenomen, waarvoor de opnamereden 'informaritis' was en die in december 1978 werden behandeld door een specialist, die in de periode januari tot en met november 1978 geen behandeling had verricht op een patiënt, die in diezelfde periode voor 'informaritis' werd opgenomen.

8.22 Geef van de patiënt met nummer 26: de opnamereden van elke opname in 1980.

8.23 Geef nummer en naam van de specialisten, die meer dan vijf maal de behandeling met code KNO6 hebben verricht.

8.24 Geef het aantal keren dat in de periode 15 september 1979 tot 16 januari 1980 het medicijn validon werd voorgeschreven.

8.25 Geef naam, adres en woonplaats van de specialisten, die sinds begin 1980 geen behandeling met code KNO6 hebben verricht.

8.26 Geef naam, adres en woonplaats van de specialisten, die sinds begin 1980 wel de behandelingen met code KNO7, KNO8 en KNO9 hebben verricht, maar niet KNO6.

8.27 Geef van elke opname in de periode 16 tot en met 23 januari 1980 in de afdeling verloskunde: nummer en naam van de patiënt, opnamereden en nummer van verpleegruimte.

8.28 Geef naam, adres en woonplaats van elke patiënt, die in februari 1981 werd opgenomen voor informaritis en die tijdens deze opname tweemaal vijf dagen het medicijn M15 kreeg voorgeschreven.

8.29 Geef van elke patiënt met bloedgroep AB: naam, adres en woonplaats en aantal dagen dat deze patiënt in maart 1980 het medicijn M12 heeft moeten gebruiken.

8.30 Geef naam, adres en woonplaats van de specialisten, die in december 1980 verantwoordelijk zijn geweest voor de opname van een patiënt uit Eindhoven en die aan deze zelfde patiënt minstens vijf maal een medicijn van de soort S05 hebben voorgeschreven.

Voor de opgaven 8.31, 8.32 en 8.33: Gegeven is een vraag en een antwoord. Ga na of het antwoord juist is. Zo nee, geef dan weer welke vraag door het gegeven antwoord wèl wordt beantwoord èn geef het correcte antwoord op de gegeven vraag.

8.31 Vraag
Geef nummer en naam van elke patiënt die in de periode 11-18 april 1980 een behandeling onderging in een behandelruimte, waarin in de tien dagen, voorafgaande aan deze behandeling een patiënt werd behandeld voor de behandeling met code BX.

Antwoord

$$\begin{aligned}\underline{get}\ (\{t/\overline{\{pnr,pnm\}}\ |\ & t\in zkh(p) \wedge \exists u\in zkh(pb)[u(pnr)=t(pnr) \wedge \\ & 800411 \leq u(dat) \leq 800418 \wedge \\ & \exists v\in zkh(pb)[v(pnr)=u(pnr) \wedge \\ & 800401 \leq v(dat) < 800411 \wedge v(bcd)=BX]]\}).\end{aligned}$$

8.32 Vraag
Geef nummer en naam van elke patiënt, die in de periode 11-18 april 1980 een behandeling onderging in een behandelruimte, waarin in de tien dagen, voorafgaande aan deze behandeling, geen patiënt werd behandeld voor een behandeling van de soort SX. (Let wel: soort is niet gelijk aan code.)

Antwoord

$$\begin{aligned}&\underline{\text{get}}\ (\{t\,/\overline{\{pnr,pnm\}}\ \mid t \in zkh(p) \wedge \exists u \in zkh(pb)\\&\quad [u(pnr) = t(pnr) \wedge 800411 \leq u(dat) \leq 800418 \wedge\\&\quad \neg(\exists v \in zkh(pb)\\&\quad [u(pnr) = v(pnr) \wedge 800401 \leq v(dat) < 800411 \wedge\\&\quad v(bcd) = SX])]\}).\end{aligned}$$

8.33 Vraag
Geef nummer, naam en woonplaats van de patiënten, die geen behandeling van soort BS13 hebben ondergaan.

Antwoord

$$\begin{aligned}&\underline{\text{get}}\ (\{t\,/\overline{\{pnr,pnm,pwpl\}}\ \mid t \in zkh(p) \wedge\\&\quad \forall u \in zkh(pb)[u(pnr) \neq t(pnr) \wedge\\&\quad (beh^{*}(u)\ \underline{\text{ext}}\ zkh(beh))(bsrt) \neq BS13\}).\end{aligned}$$

De onderhoudsopdrachten in de nu volgende opgaven uitvoeren volgens het verzamelingsmodel (en naar believen in één of meer vraagtalen).

8.34 Voeg in de volgende pb-tupels, als weergegeven in onderstaande tabel.

pnr	bcd	snr	indat	dat	duur	behr
15	DL11	13	820614	820616	250	BR11
15	DL11	13	820614	820617	120	BR10
27	DL11	13	820614	820616	130	BR11

8.35 Verander de duur van de behandeling DL11 op patiënt 15 op 16 juni 1982 in 2 uur.

8.36 Verlaag van elke behandeling van soort SDL, die in 1980 meer dan vijftig maal werd verricht, het tarief met 10%.

8.37 Verwijder het patiëntbehandelingstupel betreffende behandeling DL11 op patiënt 15 op 16 juni 1982.

8.38 Verwijder het specialistentupel van specialist 8 en alle patiëntbehandeling- en medicijnverstrekkingtupels, waarin deze specialist voorkomt.

DEEL III
HET NETWERKMODEL

9 LOGISCHE STRUCTUUR IN HET NETWERKMODEL

9.1 INLEIDING

In het eerste hoofdstuk werd er, met name in § 1.4, reeds op gewezen, dat duidelijk onderscheid dient te worden gemaakt tussen de verschillende gegevensstructuren in de verschillende fasen van een database-ontwerp. Een en ander werd schematisch weergegeven in figuur 1.5.

Bij deze figuur werd er met nadruk op gewezen, dat de opslagstructuur niet alleen afhankelijk is van de (reeds vooraf bepaalde) logische structuur, maar ook van de eisen betreffende de toegang tot de gegevens. Deze toegangseisen kunnen gemakkelijk de logische structuur 'verstoren'. Zo kan het bijvoorbeeld voorkomen, dat genormaliseerde structuren minder wenselijk zijn in de opslagfase.

Om te weten in hoeverre de logische structuur door toegangseisen wordt verstoord, dient men uiteraard logische structuur en opslagstructuur wel goed van elkaar te onderscheiden. Dit onderscheid nu (en natuurlijk daarbij ook nog het onderscheid met de fysieke structuur) is in de beschikbare database management systemen niet eenvoudig en soms zelfs moeilijk of helemaal niet aan te houden.

Wij zullen voor een bepaalde klasse van database management systemen laten zien hoe de logische structuur daarin is af te beelden en tevens hoe behandeling van queries en onderhoud in deze systemen kan plaatsvinden. Daarbij zullen diverse 'implementatiezaken' betreffende opslag van gegevens ook aan de orde (moeten) komen.

Bedoelde klasse van database management systemen is die welke gebaseerd is op het zogeheten netwerkmodel oftewel DBTG-model. Er zijn twee belangrijke redenen waarom in dit boek voor deze klasse is gekozen.

1. Een groot aantal van de beschikbare DBMS'en behoort tot deze klasse.
2. Het netwerkmodel leent zich goed om te demonstreren hoe een duidelijke logische strcutuur kan worden 'geïmplementeerd'.

Het netwerkmodel werd ontwikkeld door de zogeheten Data Base Task Group (daarvandaan DBTG-model) van het CODASYL Programming Language Committee (daarvandaan ook de benaming CODASYL-model). De benaming netwerkmodel zal verderop worden verklaard. Voor uitgebreide informatie over dit model zij verwezen naar (OLLE), (CODASYL 71), (DOROVK) en (DORSTR).

In april 1971 verscheen een rapport van de DBTG, met o.a. een voorstel voor een DDL (Data Description Language) en een DML (Data Manipulation Language). Er zijn daarna nog verschillende revisies verschenen (o.a. in 1973, 1976 en 1978), die echter het wezenlijke van de 1971-voorstellen onaangetast hebben gelaten.

In het DBTG-model worden het logische en de daaronder liggende niveaus niet altijd duidelijk gescheiden. Bij de behandeling van de diverse onderdelen zal hierop nader worden ingegaan.

De voor ons doel van belang zijnde begrippen van het DBTG-model zullen worden toegelicht met behulp van een zogeheten DBTG-schema. Dit schema wordt in de volgende paragraaf gegeven en is het DBTG-'equivalent' van de ziekenhuis-database van hoofdstuk 7.

Aangezien het de bedoeling is na te gaan hoe de logische structuur kan worden afgebeeld in een systeem volgens het DBTG-model, zullen vele van de hierbij niet ter zake doende DBTG-begrippen onbesproken blijven. Voor deze begrippen zij verwezen naar de bovengenoemde literatuur.

In dit deel over het netwerkmodel zullen wij gebruik maken van de DBMS-20 van DEC. Onder deze DBMS is op de TH Eindhoven een database (met ongeveer 100.000 recordoccurrences) volgens de structuur van hoofdstuk 7 geïmplementeerd ten behoeve van onderwijs en onderzoek.

9.2 DBTG-SCHEMA VAN EEN ZIEKENHUIS-DATABASE

Opmerkingen

- Zoals in het voorgaande reeds werd opgemerkt, is het de bedoeling onderstaand schema in de loop van de volgende paragrafen en hoofdstukken te 'ontvouwen'.
- Regels beginnend met een sterretje, zoals r. 10-12, 25-27, enz. zijn zogeheten commentaarregels. Deze behoren niet tot het schema.
- Het schema eindigt eigenlijk op r. 291. Daarna volgen nog twee subschema's.

```
SCHEMA NAME IS HOSPITAL PRIVACY LOCK FOR ADMINISTRATION IS VVJG.

AREA NAME IS STATISCH-AREA
        PRIVACY LOCK FOR PROTECTED RETRIEVAL IS RETRV.

AREA NAME IS PATIENT-AREA
        PRIVACY LOCK FOR PROTECTED RETRIEVAL IS RETRV.

AREA NAME IS OPNAME-AREA
        PRIVACY LOCK FOR PROTECTED RETRIEVAL IS RETRV.

AREA NAME IS MEDVER-AREA
        PRIVACY LOCK FOR PROTECTED RETRIEVAL IS RETRV.


*
* record beschrijving van patient
*
RECORD NAME IS P
        LOCATION MODE IS CALC USING P-NR
        DUPLICATES ARE NOT ALLOWED
        WITHIN PATIENT-AREA.

        02      P-NR            PIC 9(5).
        02      P-NM            PIC X(20).
        02      P-ADR           PIC X(20).
        02      P-WPL           PIC X(15).
        02      P-DAT           PIC X(6).
        02      P-BLOED         PIC X(2).
        02      P-RHES          PIC X(1).
        02      P-MV            PIC X(1).
*
* record beschrijving van verpleegkundige
*
RECORD NAME IS VK
        LOCATION MODE IS VIA SYS-VK
        WITHIN STATISCH-AREA.

        02      VK-NR           PIC 9(3).
        02      VK-NM           PIC X(19).
        02      VK-ADR          PIC X(18).
        02      VK-WPL          PIC X(9).
        02      VK-AFD-NR       PIC 9(2).


*
* record beschrijving van specialist
*
RECORD NAME IS SP
        LOCATION MODE IS CALC USING SP-NR
       DUPLICATES NOT ALLOWED
        WITHIN STATISCH-AREA.

        02      SP-NR           PIC 9(2).
        02      SP-NM           PIC X(19).
        02      SP-ADR          PIC X(18).
        02      SP-WPL          PIC X(9).
        02      SP-AFD-NR       PIC 9(2).
        02      SP-AB           PIC 9(2).
        02      SP-IND          PIC 9(1).
```

```
*
* record beschrijving van afdeling
*
RECORD NAME IS AFD


        LOCATION MODE IS VIA SYS-AFD
        WITHIN STATISCH-AREA.

        02      AFD-NR          PIC 9(2).
        02      AFD-NM          PIC X(20).
        02      AFD-VK-NR       PIC 9(3).
        02      AFD-SP-NR       PIC 9(2).


*
* record beschrijving van verpleegruimte
*
RECORD NAME IS VPR
        LOCATION MODE IS VIA AFD-VPR
        WITHIN STATISCH-AREA.

        02      VPR-NR          PIC 9(2).
        02      VPR-AFD-NR      PIC 9(2).
        02      VPR-AB          PIC 9(2).


*
* record beschrijving van opname
*
RECORD NAME IS OPN
        LOCATION MODE IS VIA P-OPN
        WITHIN OPNAME-AREA.

        02      OPN-P-NR        PIC 9(5).
        02      OPN-R           PIC X(40).
        02      OPN-VPR-NR      PIC 9(2).
        02      OPN-IN-DAT      PIC 9(6).
        02      OPN-UIT-DAT     PIC 9(6).
        02      OPN-SP-NR       PIC 9(2).


*
* record beschrijving van behandeling
*
RECORD NAME IS BEH
        LOCATION MODE IS CALC USING BEH-CD
        DUPLICATES NOT ALLOWED
        WITHIN STATISCH-AREA.

        02      BEH-CD          PIC 9(4).
        02      BEH-NM          PIC X(50).
        02      BEH-SRT         PIC X(3).
        02      BEH-TAR         PIC 9(3).
```

```
*
* record beschrijving van patient behandeling
*
RECORD NAME IS PATBEH
        LOCATION MODE IS VIA OPN-PATBEH
        WITHIN OPNAME-AREA.

        02      PB-P-NR         PIC 9(5).
        02      PB-BEH-CD       PIC 9(4).
        02      PB-SP-NR        PIC 9(2).
        02      PB-DAT          PIC 9(6).
        02      PB-DUUR         PIC 9(3).
        02      PB-BEHR         PIC X(4).


*
* recordbeschrijving van medicijn
*
RECORD NAME IS MED
        LOCATION MODE IS CALC USING MED-CD
        DUPLICATES NOT ALLOWED
        WITHIN STATISCH-AREA.

        02      MED-CD          PIC X(6).
        02      MED-NM          PIC X(20).
        02      MED-SRT         PIC X(4).


*
* record beschrijving van cummulatieve recepten
*
RECORD NAME IS MEDVER
        LOCATION MODE IS VIA P-MEDVER
        WITHIN MEDVER-AREA.

        02      MV-MED-CD       PIC X(6).
        02      MV-P-NR         PIC 9(5).
        02      MV-SP-NR        PIC 9(2).
        02      MV-DATUM        PIC 9(6).
        02      MV-DUUR         PIC 9(2).
        02      MV-DAG          PIC 9(1).
        02      MV-AANT         PIC 9(1).


SET NAME IS SYS-AFD
                MODE IS CHAIN
                ORDER IS SORTED
                DUPLICATES NOT ALLOWED
        OWNER IS SYSTEM
        MEMBER IS AFD
                MANDATORY AUTOMATIC
                ASCENDING KEY IS AFD-NR.


SET NAME IS SYS-VK
                MODE IS CHAIN
                ORDER IS SORTED
                DUPLICATES NOT ALLOWED
        OWNER IS SYSTEM
        MEMBER IS VK
                MANDATORY AUTOMATIC
                ASCENDING KEY IS VK-NR.
```

```
SET NAME IS SYS-SP
                MODE IS CHAIN
                ORDER IS SORTED
                DUPLICATES NOT ALLOWED
        OWNER IS SYSTEM
        MEMBER IS SP
                MANDATORY AUTOMATIC
                ASCENDING KEY IS SP-NR.

SET NAME IS SYS-P
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS SYSTEM
        MEMBER IS P
                MANDATORY AUTOMATIC.

SET NAME IS SYS-MED
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS SYSTEM
        MEMBER IS MED
                MANDATORY AUTOMATIC.

SET NAME IS SYS-BEH
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS SYSTEM
        MEMBER IS BEH
                MANDATORY AUTOMATIC.

SET NAME IS AFD-VPR
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS AFD
        MEMBER IS VPR
                MANDATORY AUTOMATIC
                SET OCCURRENCE SELECTION THRU
                        LOCATION MODE OF OWNER
                        USING AFD-NR.

SET NAME IS VPR-OPN
                MODE IS CHAIN
                ORDER IS SORTED
        OWNER IS VPR
        MEMBER IS OPN
                AUTOMATIC MANDATORY
                LINKED TO OWNER
                ASCENDING KEY IS OPN-IN-DAT
                        DUPLICATES ARE LAST ALLOWED
                SET OCCURRENCE SELECTION THRU
                        CURRENT OF SET.
```

```
SET NAME IS AFD-VK
                    MODE IS CHAIN
                    ORDER IS ALWAYS LAST
          OWNER IS AFD
          MEMBER IS VK
                    MANDATORY AUTOMATIC
                    SET OCCURRENCE SELECTION IS THRU
                              LOCATION MODE OF OWNER
                              USING AFD-NR.


SET NAME IS AFD-SP
                    MODE IS CHAIN
                    ORDER IS ALWAYS LAST
          OWNER IS AFD
          MEMBER IS SP
                    MANDATORY AUTOMATIC
                    SET OCCURRENCE SELECTION IS THRU
                              LOCATION MODE OF OWNER
                              USING AFD-NR.


SET NAME IS SP-OPN
                    MODE IS CHAIN
                    ORDER IS ALWAYS LAST
          OWNER IS SP
          MEMBER IS OPN
                    AUTOMATIC MANDATORY
                    LINKED TO OWNER
                    SET OCCURRENCE SELECTION IS THRU
                              LOCATION MODE OF OWNER.


SET NAME IS SP-PATBEH
                    MODE IS CHAIN
                    ORDER IS SORTED
          OWNER IS SP
          MEMBER IS PATBEH
                    MANDATORY AUTOMATIC
                    LINKED TO OWNER
                    ASCENDING KEY IS PB-BEH-CD
                              DUPLICATES ARE LAST ALLOWED
                    SET OCCURRENCE SELECTION IS THRU
                              LOCATION MODE OF OWNER.


SET NAME IS P-OPN
                    MODE IS CHAIN
                    ORDER IS SORTED
          OWNER IS P
          MEMBER IS OPN
                    MANDATORY AUTOMATIC
                    ASCENDING KEY IS OPN-SP-NR
                              DUPLICATES ARE LAST ALLOWED
                    SET OCCURRENCE SELECTION IS THRU
                              LOCATION MODE OF OWNER.
```

```
SET NAME IS P-PATBEH
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS P
        MEMBER IS PATBEH
                MANDATORY AUTOMATIC
                SET OCCURRENCE SELECTION IS THRU
                        LOCATION MODE OF OWNER.


SET NAME IS OPN-PATBEH
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS OPN
        MEMBER IS PATBEH
                MANDATORY AUTOMATIC
                SET OCCURRENCE SELECTION IS THRU
                        CURRENT OF SET.


SET NAME IS P-MEDVER
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS P
        MEMBER IS MEDVER
                MANDATORY AUTOMATIC
                SET OCCURRENCE SELECTION IS THRU
                        LOCATION MODE OF OWNER.


SET NAME IS BEH-PATBEH
                MODE IS CHAIN
                ORDER IS ALWAYS LAST
        OWNER IS BEH
        MEMBER IS PATBEH
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                SET OCCURRENCE SELECTION IS THRU
                        LOCATION MODE OF OWNER.


SET NAME IS SP-MEDVER
                MODE IS CHAIN
                ORDER IS SORTED
        OWNER IS SP
        MEMBER IS MEDVER
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS MV-MED-CD
                        DUPLICATES ARE LAST ALLOWED
                SET OCCURRENCE SELECTION IS THRU
                        LOCATION MODE OF OWNER.
```

```
SET NAME IS MED-MEDVER
                MODE IS CHAIN
                ORDER IS SORTED
        OWNER IS MED
        MEMBER IS MEDVER
                MANDATORY AUTOMATIC
                LINKED TO OWNER
                ASCENDING KEY IS MV-P-NR
                        DUPLICATES ARE LAST ALLOWED
                SET OCCURRENCE SELECTION IS THRU
                        LOCATION MODE OF OWNER.


SUB-SCHEMA NAME IS HOSPIT-SUB01
        PRIVACY LOCK GVJV.
AREA SECTION.
        COPY ALL AREAS.
RECORD SECTION.
        COPY ALL RECORDS.
SET SECTION.
        COPY ALL SETS.

SUB-SCHEMA NAME IS HOSPIT-PRAKT
        PRIVACY LOCK IS PRAKT.
AREA SECTION.
        COPY PATIENT-AREA,OPNAME-AREA.
RECORD SECTION.
        01      P.
                02      P-NR.
                02      P-WPL.
        01      OPN.
                02      OPN-P-NR.
                02      OPN-R.
                02      OPN-IN-DAT.
                02      OPN-UIT-DAT.
SET SECTION.
        COPY P-OPN.

END-SCHEMA.
```

9.3 RECORDTYPE EN SETTYPE

Een DDL, volgens het DBTG-model, werkt met drie basisbegrippen: area, recordtype en settype.

Een *area* is een deel van het (achtergrond)geheugen dat beschikbaar is voor de database. Dit begrip is van geen belang voor de weergave van de betekenis van de gegevens, dus voor de logische structuur, maar wel voor de realisatie van de toegangseisen, dus voor de opslagstructuur. Met geschikt gekozen area's is het mogelijk efficiënt onderhoud en gebruik van de gegevens te bewerkstelligen. In ons schema (r. 2-9) is sprake van vier area's: STATISCH-AREA, PATIENT-AREA, OPNAME-AREA en MEDVER-AREA.

Een *record*, of ook dikwijls *recordtype*, is eigenlijk een bijzonder geval van het in hoofdstuk 3 genoemde tupeltype. Het bijzondere zit in het feit, dat niet of nauwelijks mogelijkheden aanwezig zijn om tupel- en tabelconstraints te definiëren. Alleen via de zogeheten LOCATION MODE IS CALC (zie verderop) is het mogelijk op een impliciete wijze aan te geven dat een groep identifiers uniek identificerend is. Evenals voor het recordtype van Pascal geldt dus ook voor het DBTG-recordtype dat het een bijzonder geval is van een tupeltype.

De tien objecten van hoofdstuk 7 vinden wij in het schema van § 9.2 terug in evenzovele recordbeschrijvingen, beginnend met "RECORD NAME IS". De LOCATION MODE-clausule zal, zoals gezegd, verderop worden behandeld.

WITHIN PATIENT-AREA (r. 16) betekent, dat de recordoccurrences van het desbetreffende recordtype moeten worden opgeslagen in de area PATIENT-AREA.

Bij de beschrijving van de data-items (attributen) van een recordtype wordt de COBOL-aanpak gevolgd met level numbers (02 en eventueel ook hogere) en PIC(TURE)-clausules. Zo'n PIC-clausule is in feite een type-definitie voor een attribuut. Zo is (r.17)

```
"02        P-NR                 PIC        9(5)"
```

het netwerkequivalent van

```
"P-NR : {0...99999}"
```

en (r.18)

```
"02        P-NM                 PIC        X(20)"
```

het netwerkequivalent van

```
"P-NM : charstring 20".
```

Opmerking
De attribuutnamen van hoofdstuk 7 konden niet letterlijk worden overgenomen, omdat in het onderhavige DBMS geëist wordt dat elke attribuutnaam uniek is binnen het totale schema. □

Met een *settype* wordt beoogd een verband tussen twee recordtypen vast te leggen en wel op de volgende wijze. Een van deze twee recordtypen is (binnen het gegeven settype) het zogeheten *owner recordtype*, of kortweg *owner*, het andere recordtype is het zogeheten *member recordtype* of kortweg *member*. Bij elke occurrence van de owner horen nu n ($n \geqq 0$) occurrences van de member, met dien verstande dat een occurrence van de member bij precies één occurrence van de owner hoort.

Opmerking
Bij de bespreking van het begrip membership in hoofdstuk 12 zal blijken dat de voorafgaande zin volgens de geldende regels van het netwerkmodel moet luiden: bij elke occurrence van de owner horen nu n ($n \geqq 0$) occurrences van de member, met dien verstande dat een occurrence van de member bij *hoogstens* een occurrence van de owner hoort.

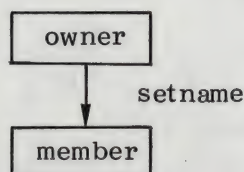
Wij gaan in dit en de volgende twee hoofdstukken uit van de eerste versie, dus zonder "hoogstens". □

In het schema worden 19 settypes gedefinieerd. De naam van de eerste zes settypes begint met "SYS". Deze eerste zes settypes zullen wij voorlopig buiten beschouwing laten. Van de overige dertien nu enkele voorbeelden.

Voorbeeld 9.1

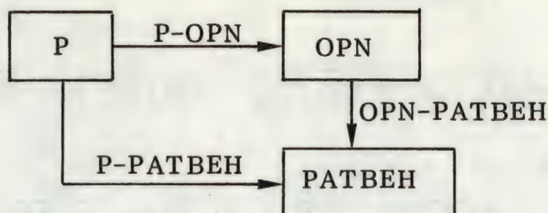
Het settype P-OPN heeft als owner het recordtype P (r. 230) en als member het recordtype OPN (r. 231). In het settype P-PATBEH komt P weer als owner voor (r. 240), maar nu is PATBEH member (r. 241). In het settype OPN-PATBEH is OPN owner (r. 248) en PATBEH member (r. 249). □

Een settype kan als volgt, met een zogeheten Bachman-diagram, in beeld worden gebracht.



Voorbeeld 9.2

Het Bachman-diagram voor de settypen van voorbeeld 9.1 is als volgt:



□

Eenzelfde recordtype kan in meer dan één settype optreden als owner en/of member. Hierdoor is een 'netwerk' van verbanden mogelijk tussen de recordtypen. Daarvandaan de naam netwerkmodel. Hierbij geldt overigens een merkwaardige beperking: een recordtype kan binnen een settype niet tegelijk owner en member zijn.

Voor de volledigheid moet worden opgemerkt, dat volgens het DBTG-voorstel ook settypen mogelijk zijn met meer dan één membertype. In de praktijk wordt hier vrijwel geen gebruik van gemaakt, omdat het

- eigenlijk een overbodige faciliteit is (zeker gezien vanuit de logische structuur);
- in verschillende netwerk-DBMS-en niet geïmplementeerd is (dit geldt ook voor DBMS-20 van DEC).

Naar analogie van recordoccurrences bij recordtypen spreekt men bij settypen van *setoccurrences*. Een setoccurrence van een bepaald settype is een verzameling, bestaande uit precies één owneroccurrence èn de bij deze owneroccurrence behorende memberoccurrences.

Een setoccurrence, die alleen bestaat uit een owneroccurrence, heet een *lege setoccurrence*. Bij de definitie van een settype moet worden vastgelegd hoe de records per setoccurrence moeten worden 'samengevoegd' en in welke ordening deze samenvoeging moet plaatsvinden. Voor de samenvoeging zijn twee manieren beschikbaar, namelijk met behulp van

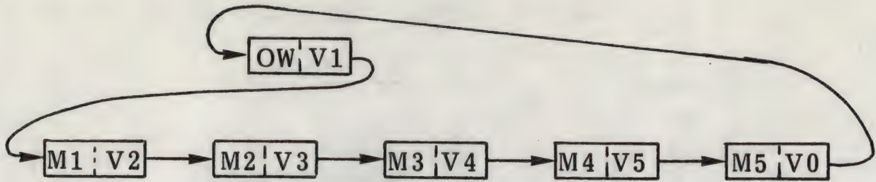
- een kettingstructuur, of
- een pointer-array.

Bij gebruik van een ketting per setoccurrence wordt in het owner-record een (adres)verwijzing opgenomen naar het 'eerste' bijbehorende member-record. Dit 'eerste' heeft een verwijzing naar het 'tweede', enz. Het 'laatste' member-record heeft een verwijzing (terug) naar de owner.

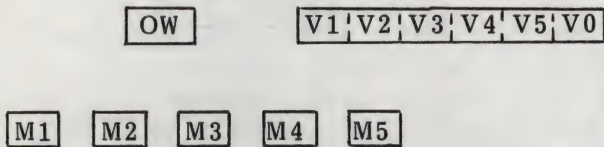
Bij gebruik van een pointer-array staan genoemde (adres)verwijzingen niet in de records maar in een apart array.

Beide manieren staan onderstaand in figuur 9.1 schematisch weergegeven voor een setoccurrence met vijf memberoccurrences.

Met ketting:



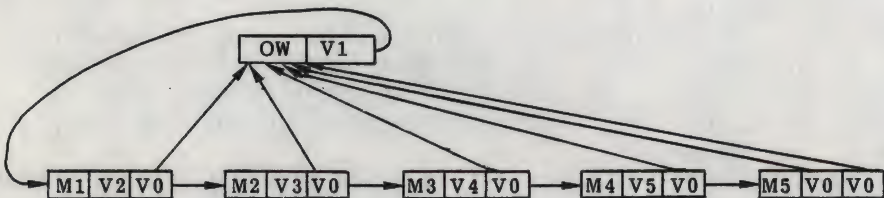
Met pointer-array:



Figuur 9.1 Schematische weergave van ketting en pointer-array.
OW: owneroccurrence
M1,...,M5: memberoccurrences
V1,...,V5: adresverwijzing naar volgende memberoccurrence
V0: adresverwijzing naar owneroccurrence.

In het schema van § 9.2 komt uitsluitend de kettingstructuur voor (MODE IS CHAIN) (r. 128, 136, enz.).

Als men vanuit een memberoccurrence de bijbehorende owneroccurrence wil benaderen, zit er (zonder verdere voorzieningen) niets anders op dan de hele ketting 'af te lopen' totdat men bij deze owneroccurrence is 'aangeland'. Er is echter een mogelijkheid om een 'kortsluiting' tot stand te brengen met behulp van de clausule "LINKED TO OWNER". In het schema komt dit bijvoorbeeld voor in regel 222 in de beschrijving van het settype SP-PATBEH. Dit laatste betekent dat er bij elke occurrence van PATBEH een adresverwijzing is opgenomen naar de SP-occurrence, dat owner is van deze PATBEH-occurrence. Zie figuur 9.2.



Figuur 9.2 Ketting èn linked to owner (vgl. fig. 9.1).

Uit het bovenstaande blijkt dat het netwerkmodel toch een zekere 'hang' heeft naar hiërarchische structuren, omdat de toegang van owner naar member (via ketting of pointer-array) verplicht is en de omgekeerde toegang facultatief. Dit betekent, in geval alleen de laatste toegang nodig is, een onnodig stuk onderhoud voor de eerste, niet benodigde, toegang.

De ordening binnen een setoccurrence kan zijn
- SORTED of
- FIRST, LAST, NEXT, PRIOR

In het geval "ORDER IS SORTED" wordt enkele regels eronder met ASCENDING/DESCENDING KEY-clausules aangegeven hoe de memberrecords per setoccurrence gesorteerd zijn.

Met FIRST, respectievelijk LAST wordt bedoeld, dat elke nieuwe occurrence van het memberrecordtype als eerste respectievelijk laatste in de setoccurrence (ketting of pointer-array) moet worden opgenomen.

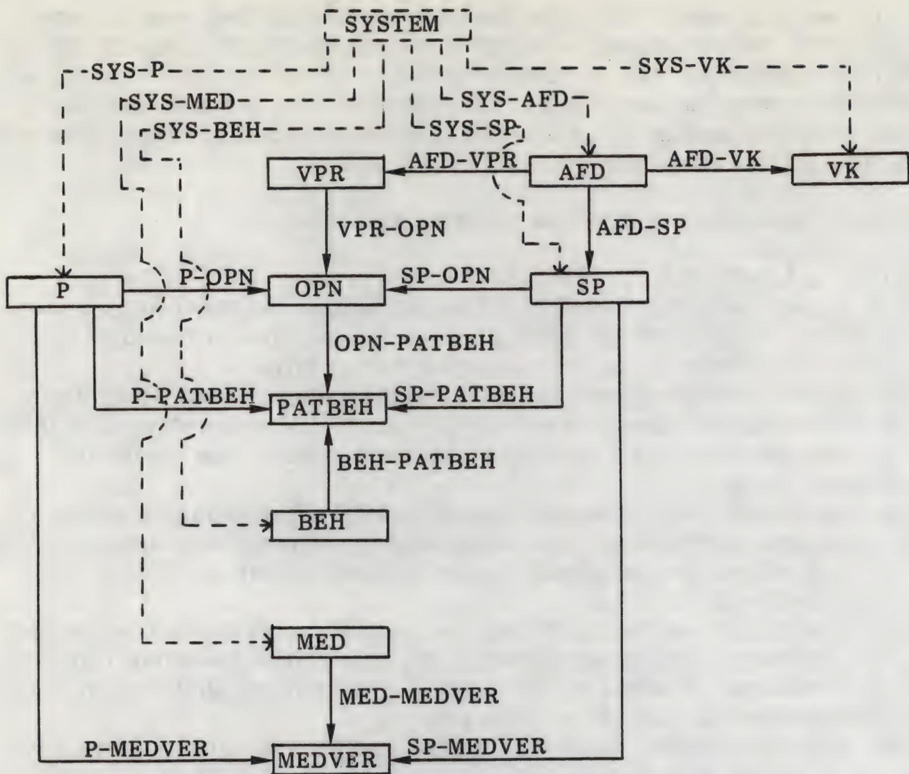
In geval van NEXT respectievelijk PRIOR vindt opname plaats volgend op respectievelijk voorafgaand aan de recordoccurrence, die de current of settype aangeeft (voor dit laatste begrip zie het volgende hoofdstuk, § 10.1).

Met behulp van setoccurrences kunnen de recordoccurrences van het memberrecordtype *groepsgewijze* ('sequentieel') worden opgeslagen en benaderd. Hierbij zal het aantal groepen gelijk zijn aan het aantal occurrences van het ownertype.

Om *alle* occurrences van een recordtype als *een* groep (sequentieel) te kunnen benaderen, moet een apart settype worden gedefinieerd, met genoemd recordtype als member en met precies één setoccurrence. Aangezien de owner 'er niet toe doet', wordt deze 'taak' door het DBMS 'overgenomen' en zal de owner de standaardnaam 'SYSTEM' krijgen. Zie de eerste zes settypen in het schema. Een settype met SYSTEM als owner wordt ook wel een *singular set* genoemd.

Het volledige Bachman-diagram, behorende bij het schema van § 9.2 ziet eruit als in figuur 9.3 (singular sets zijn gestippeld weergegeven).

Als we het Bachman-diagram in figuur 9.3 vergelijken met het diagram met sterfuncties in figuur 8.1, valt onmiddellijk de grote gelijkenis op. Deze gelijkenis is als volgt te verklaren.
- Stilzwijgend hebben wij verondersteld, dat voor elk object in de ziekenhuis-database van hoofdstuk 7 een recordtype zal worden gedefinieerd bij weergave in het netwerkmodel.
  Hiermee blijft de 'normaalvorm' behouden. Let wel: het netwerkmodel spreekt zich niet uit over (gewenste) normalisatie, kent dit begrip zelfs niet, maar er is natuurlijk niets tegen om ook in het netwerkmodel met een genormaliseerde database te werken. Het netwerkmodel leent zich daar trouwens uitstekend voor door



Figuur 9.3 Bachman-diagram van de ziekenhuis-database.

-- enerzijds de *mogelijkheid* tot definitie van settypen 'tussen' elk willekeurig tweetal recordtypen
-- en anderzijds de *beperking* dat settypen alleen maar gebruikt mogen worden voor de weergave van één-op-veel (1:n) verbanden tussen owner en member.

Met dit laatste wordt bedoeld (vergelijk § 5.4), dat bij een owner-occurrence een aantal van n ($n \geq 0$) memberoccurrences horen èn bij een memberoccurrence precies één owneroccurrence. In het verzamelingsmodel kan in een genormaliseerd databasetype een veel-op-veel (n:m) verband tussen twee objecten alleen worden weergegeven met behulp van een apart object met bijbehorend tabeltype en twee ssr's. Evenzo kan een (n:m) verband tussen twee recordtypen in een 'genormaliseerd' netwerkschema alleen worden weergegeven met behulp van een zogeheten link-record en twee settypen, die beide dit link-record als member hebben.
Analoge beweringen gelden voor een (n1:n2:n3) verband tussen

drie recordtypen. Zie bijvoorbeeld het n1:n2:n3 verband tussen de recordtypen P, SP en VPR, weergegeven door middel van het link-record OPN.
Elke combinatie van een sterfunctie met een pijl in figuur 8.1 is de weergave van een (n:1) verband tussen twee objecten (een pijl stelt in feite het ext-gedeelte voor). Wil men dit verband behouden in een netwerkschema, dan kan men dit eenvoudig waarmaken met behulp van een settype (en in het diagram met een 'omgekeerde' pijl). Wij vinden alle combinaties van een sterfunctie met een pijl in figuur 8.1 'terug' in figuur 9.3 in de vorm van settypen. (Alleen de combinaties vk* met pijl naar vk en spec* met pijl naar sp vinden wij in figuur 9.3 niet terug. Hier is namelijk in feite sprake van een (1:1) verband en daarvan is de weergave met behulp van een settype wel technisch mogelijk, maar niet erg zinvol.)
Er zij op gewezen dat bij deze gelijkenis tussen de figuren 8.1 en 9.3 een belangrijk verschil tussen het verzamelingsmodel en het netwerkmodel niet uit het oog moet worden verloren, namelijk dat een ssr geen adequaat equivalent heeft in het netwerkmodel. Zo kan bijvoorbeeld via het schema niet geëist worden, dat elke waarde van PB-P-NR in de recordoccurrences van PATBEH, ook voorkomt als waarde van P-NR in een occurrence van P. Dit is namelijk niet hetzelfde als de eis dat tussen de recordtypen P en PATBEH een (1:n) verband moet bestaan, hetgeen in het netwerkmodel wel is weer te geven met behulp van een settype.

Tot nu toe is bij het onderwerp record-benadering alleen de groepsgewijze benadering (via setoccurrences) aan bod geweest. Er is echter ook de mogelijkheid om één record, los van andere records, apart te benaderen, mits in de definitie van het desbetreffende recordtype daartoe de juiste voorzieningen zijn getroffen en wel door geschikte keuze van de LOCATION MODE-clausule.

Bij deze clausule bestaan de volgende opties.

LOCATION MODE IS

a. DIRECT
b. CALC USING <identifier(s)>
c. VIA <set-name> SET.

Aparte benadering van één enkel record is mogelijk als optie a of b van de LOCATION MODE wordt gebruikt in de beschrijving van het desbetreffende recordtype. Bij optie c is deze directe benadering niet mogelijk. Met deze optie c wordt bedoeld aan te geven dat elke occurrence van het desbetreffende recordtype 'zo gunstig mogelijk' met het oog op het genoemde settype moet worden opgeslagen. Zo zijn de afdelingsrecords niet direct te benaderen, omdat voor deze records geldt LOCATION MODE IS VIA SYS-AFD (r. 54). Andere voorbeelden in r. 29, 64, 73, 96 en 118.

Bij optie a is wel directe benadering mogelijk, mits men kan beschikken over de zogeheten databasekey. Een *databasekey* identificeert elke recordoccurrence uniek binnen de totale database. Het is een item, dat door het DBMS wordt beheerd en dan ook *niet* in het schema *hoeft* te worden opgenomen. Wel *mag* een gebruiker een veld definiëren dat een databasekey-waarde kan bevatten. Wij komen hierop terug bij de behandeling van currency-problemen (§ 10.1).

In optie b is CALC een afkorting van CALCULATION, aangevend dat een berekening moet plaatsvinden om de plaats (LOCATION) van een record vast te leggen. Deze berekening zal gebruik maken van (USING) de door de gebruiker opgegeven identifier(s). De toevoeging DUPLICATES NOT ALLOWED betekent dat elke mogelijke waarde van genoemde identifier(s) niet meer dan eenmaal mag voorkomen, dus dat genoemde identifier(s) een sleutel bevatten van het desbetreffende recordtype. Met deze toevoeging is optie b gelijk aan het geval van sleutelconversie met behulp van een hashing-functie in de klassieke bestandsorganisatie (LUNREM, hfdst.6). In het schema van § 9.2 komt de LOCATION MODE IS CALC alleen voor met de toevoeging DUPLICATES NOT ALLOWED.

In deze paragraaf zijn nu alle elementen van de record-definities in het schema van § 9.2 ter sprake gekomen; van de set-definities zijn nog niet behandeld

- de zogeheten membership-clausule, die in het schema in de vorm AUTOMATIC MANDATORY (of andersom) voorkomt, en
- de SET SELECTION-clausule.

De begrippen membership en SET SELECTION komen in het laatste hoofdstuk aan bod.

Uit het voorgaande mogen al wel de volgende conclusies duidelijk zijn.

- Een objectkarakterisering laat zich goed weergeven in het netwerkmodel.
- Het netwerkmodel kent geen tupelconstraints, tabelconstraints (behoudens impliciete sleutel-definitie) of databaseconstraints (behoudens 'onvolledige' ssr's). Er kunnen dan ook alleen maar (vrijwel) constraint-loze typen in worden weergegeven.
- In het netwerkmodel is men bij de weergave van de logische structuur verplicht veel aandacht te besteden aan zaken betreffende de opslagstructuur (LOCATION MODE- en AREA-clausules bij de beschrijving van recordtypes; MODE IS CHAIN, ORDER-clausule en de nog te bespreken SET SELECTION-clausule bij de beschrijving van settypes).

OPGAVEN

9.1 Ontwerp een schema met bijbehorend Bachman-diagram voor
a. databasetype DT-HANDEL-1 (voorbeeld 6.1) en voor databasetype DT-HANDEL-2 (voorbeeld 6.4);
b. het databasetype van opgave 6.3.

9.2 Deze opgave heeft betrekking op het schema van § 9.2. Verder wordt uitgegaan van de volgende aantallen occurrences per recordtype:

recordtype	aantal occurrences
P	10000
VK	140
SP	30
AFD	10
VPR	40
OPN	20000
BEH	100
PATBEH	40000
MED	200
MEDVER	50000

a. Geef het aantal recordoccurrences per area.
b. Geef commentaar op de volgende beweringen.
  b1. De occurrences van alle recordtypen, die als owner in een settype voorkomen, kunnen direct worden benaderd.
  b2. Een setoccurrence van het settype SYS-AFD bevat gemiddeld een recordoccurrence.
  b3. Een setoccurrence van het settype P-PATBEH bevat gemiddeld een occurrence van recordtype P en vier occurrences van recordtype PATBEH.
  b4. Een setoccurrence van het settype P-PATBEH kan leeg zijn.
c. Kan "LINKED TO OWNER" van belang zijn voor het settype P-OPN?

10 RETRIEVAL IN HET NETWERKMODEL; CURRENCY

10.1 FIND- EN GET-OPDRACHT; CURRENCY

In het DBTG-model werkt men met een host language (vgl. § 1.3). In ons geval zal dat COBOL zijn. Deze taal is daartoe uitgebreid met enkele DML-opdrachten voor
- onderhoud (maintenance), en
- gebruik (retrieval.

In het volgende hoofdstuk zal onderhoud worden behandeld. In dit hoofdstuk beperken wij ons tot retrieval.

Voor retrieval zijn twee opdrachten beschikbaar, namelijk de FIND- en de GET-opdracht. De FIND-opdracht dient om de plaats van een recordoccurrence in de database te bepalen; met de GET-opdracht wordt een recordoccurrence ingelezen in het werkgeheugen.
   In een FIND-opdracht moet altijd het volgende aanwezig zijn:

FIND <record-selection-expression>.

Wij zullen 'rse' gebruiken als afkorting van 'record-selection-expression'. Wij zullen in dit boek alleen de volgende drie soorten rse's bespreken, namelijk die welke betrekking hebben op

rse1: directe toegang tot een recordoccurrence
rse2: seriële toegang binnen een setoccurrence
rse3: directe toegang tot de owner binnen een setoccurrence.

De 'nummering' rse1, rse2 en rse3 geldt enkel voor dit boek; in de meeste boeken en manuals wordt de nummering van officiële DBTG-specificaties aangehouden.

De vorm van de drie rse's is als volgt:

rse1: <recordname> RECORD

rse2: { NEXT | PRIOR | FIRST | LAST } RECORD OF <setname> SET

rse3: OWNER OF <setname> SET

met de volgende betekenis (waarbij er voorlopig van uitgegaan wordt, dat alle opdrachten uitvoerbaar zijn).

Ad rse1
Deze kan alleen worden toegepast op recordtypen met LOCATION MODE IS CALC. Door aan de USING-identifier(s) de juiste waarde te geven kan verder met een FIND-opdracht de plaats van het gewenste record worden gevonden.

Voorbeeld 10.1 (op het schema van § 9.2)

Met

```
MOVE 10 TO P-NR
FIND P RECORD
```

wordt de plaats bepaald van het patiënt-record met patiëntnummer 10. □

Ad rse2
In een FIND-opdracht moet een van de vier tussen accoladen genoemde mogelijkheden worden gebruikt.

Met {NEXT / PRIOR / FIRST / LAST} wordt de plaats bepaald van

het {volgende / vorige / eerste / laatste} memberrecord

van een bepaalde setoccurrence van het <setname> type.

Voorbeeld 10.2 (op het schema van § 9.2)

Met

```
MOVE 10 TO P-NR
FIND P RECORD
FIND FIRST RECORD OF P-OPN SET
FIND NEXT  RECORD OF P-OPN SET
```

wordt achtereenvolgens de plaats bepaald van:

1. het patiënt-record met patiëntnummer 10;
2. het eerste opname-record in de setoccurrence met genoemd patiënt-record als owner;
3. het tweede opname-record in dezelfde setoccurrence. □

Ad rse3
Met een FIND-opdracht wordt de plaats bepaald van het owner-record van een setoccurrence.

Voorbeeld 10.3 (op het schema van § 9.2)

Met

```
MOVE 10 TO P-NR
FIND P RECORD
FIND FIRST RECORD OF P-OPN SET
FIND OWNER OF SP-OPN SET
FIND NEXT RECORD OF P-OPN SET
FIND OWNER OF SP-OPN SET
```

wordt achtereenvolgens de plaats bepaald van:

1. het patiënt-record met patiëntnummer 10;
2. het eerste opname-record in de setoccurrence met genoemd patiënt-record als owner;
3. het specialist-record, dat owner is van de setoccurrence waarin het opname-record sub 2 als member voorkomt;
4. het tweede opname-record in de setoccurrence, genoemd sub 2;
5. het specialist-record, dat owner is van de setoccurrence waarin het opname-record sub 4 als member voorkomt. □

Tot zover (hopelijk) geen problemen.
Wat echter te denken van het volgende voorbeeld?

Voorbeeld 10.4 (op het schema van § 9.2)

Gegeven het programmadeel

```
MOVE 10 TO P-NR
FIND P RECORD
FIND FIRST RECORD OF P-OPN SET
FIND OWNER OF SP-OPN SET
FIND FIRST RECORD OF SP-OPN SET
FIND NEXT RECORD OF P-OPN SET
```

Op 'het eerste gezicht' lijkt het dat met de laatste FIND hetzelfde opname-record wordt gevonden als opname-record sub 4 van voorbeeld 10.3. Dit is echter niet juist. □

Het probleem met het laatste voorbeeld is, dat wij niet precies het volledige effect van een FIND-opdracht kennen. Daarvoor zullen wij het zogeheten currency-probleem moeten bespreken.

Daarbij moeten wij gebruik maken van de reeds eerder genoemde databasekey. Zoals gezegd wordt met de databasekey (de plaats van) elke recordoccurrence in de database uniek vastgelegd. In principe hoeft de gebruiker niets van de databasekey of zelfs maar van het

bestaan daarvan te weten. Voor een goed begrip van het currency-probleem moeten wij er echter wel even aandacht aan besteden.

Wij zullen aannemen dat (zoals trouwens in verschillende implementaties ook het geval is) de databasekey-waarden bestaan uit een volgnummer per recordtype, gevolgd door een volgnummer per recordoccurrence binnen een recordtype. In onderstaand voorbeeld 10.5 staan deze waarden genoteerd in de kolom 'databasekey'.

Voorbeeld 10.5

Gegeven is het volgende gedeelte van een schema, met weglating van voor dit en volgende voorbeelden niet ter zake doende AREA-clausules.

```
SCHEMA NAME IS VB10

RECORD NAME IS A
        LOCATION MODE IS CALC USING A1
        DUPLICATES NOT ALLOWED.
        02    A1    PIC    99.
        02    A2    PIC    99.

RECORD NAME IS B
        LOCATION MODE IS CALC USING B1
        DUPLICATES NOT ALLOWED.
        02    B1    PIC    99.
        02    B2    PIC    99.

RECORD NAME IS C
        LOCATION MODE IS VIA AC
        02    C1    PIC    99.
        02    C2    PIC    99.
        02    C3    PIC    99.

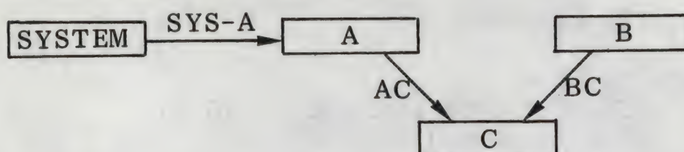
SET NAME IS SYS-A
        MODE IS CHAIN
        ORDER IS SORTED
   OWNER IS SYSTEM
   MEMBER IS A
        AUTOMATIC MANDATORY
        ASCENDING KEY IS A1.

SET NAME IS AC
        MODE IS CHAIN
        ORDER IS SORTED
   OWNER IS A
   MEMBER IS C
        AUTOMATIC MANDATORY
        ASCENDING KEY IS C2
        SET OCCURRENCE SELECTION IS THRU
           CURRENT OF SET.
```

```
SET NAME IS BC
        MODE IS CHAIN
        ORDER IS ALWAYS LAST
   OWNER IS B
   MEMBER IS C
        AUTOMATIC MANDATORY
        SET OCCURRENCE SELECTION IS THRU
           CURRENT OF SET.

END-SCHEMA
```

Het bij bovenstaand schema behorende Bachman-diagram is



In onderstaande tabel worden de recordoccurrences gegeven met bijbehorende waarden van de databasekey.

Recordtype A

A1	A2	data-base-key
1	5	1.12
2	7	1.01
6	3	1.05
7	5	1.02
10	2	1.10
13	3	1.07
18	4	1.06

Recordtype B

B1	B2	data-base-key
5	6	2.08
9	1	2.16
12	2	2.09
16	7	2.05
23	7	2.01
26	2	2.03
29	6	2.02
41	8	2.10
57	5	2.11

Recordtype C

C1	C2	C3	data-base-key
1	29	78	3.07
1	9	15	3.02
1	12	52	3.13
1	23	40	3.32
1	16	77	3.01
2	23	11	3.29
2	5	51	3.03
2	12	05	3.04
2	57	54	3.05
7	9	08	3.10
7	26	75	3.11
7	12	56	3.12
10	9	77	3.09
10	29	32	3.15
10	26	91	3.23
10	5	52	3.17
10	57	68	3.19
10	23	13	3.22
18	29	15	3.28

'Gaten' in een volgnummerserie van de databasekey worden verondersteld veroorzaakt te zijn door verwijdering van recordoccurrences.

Verder worden nog de volgende veronderstellingen gemaakt:
- Elk C-record zal binnen settype AC behoren tot de setoccurrence waarvan de owner voor A1 dezelfde waarde heeft als C voor C1.
- Elk C-record zal binnen settype BC behoren tot de setoccurrence waarvan de owner voor B1 dezelfde waarde heeft als C voor C2.
- Per setoccurrence van settype BC zijn de C-records 'toevallig' naar oplopende databasekey-waarde geordend. □

Zoals bekend dient de FIND-opdracht om de plaats van een recordoccurrence in de database te bepalen. Dit kan gevoegelijk zo worden voorgesteld dat de FIND-opdracht de databasekey-waarde levert van de recordoccurrence, die wordt gespecificeerd met de rse.

Voorbeeld 10.6

(zie gegevens in voorbeeld 10.5)

Met het programmadeel:

```
MOVE 7 TO A1
FIND A RECORD
FIND FIRST RECORD OF AC SET
FIND OWNER OF BC SET
FIND NEXT RECORD OF AC SET
FIND OWNER OF BC SET
```

komen achtereenvolgens de volgende databasekey-waarden beschikbaar:
1.02; 3.10; 2.16; 3.12; 2.09. (Men ga dit na!) □

Een netwerk-DBMS kent *per programma* verschillende zogeheten *currency-indicators* (systeemgrootheden) en wel een indicator voor het onderhanden zijnde programma, een indicator voor elk recordtype en een indicator voor elk settype. Ook is er voor elke area een indicator, maar deze kunnen wij hier gevoegelijk buiten beschouwing laten.

De indicator voor het programma, de zogeheten *current of run-unit*, bevat het resultaat van de laatst uitgevoerde FIND-opdracht, in de vorm van de databasekey-waarde van het record, dat 'onderwerp' was van deze FIND.

De indicator van recordtype A, de zogeheten *current of recordtype A*, bevat de databasekey-waarde die resulteerde uit de laatste FIND-opdracht op recordtype A.

De indicator van settype AC, de zogeheten *current of settype AC*, bevat de databasekey-waarde, die resulteerde uit de laatste FIND-opdracht op het owner- of memberrecordtype van settype AC.

Door een FIND-opdracht, waarbij R het record is dat bepaald is

door de rse, krijgen de volgende currency-indicators de waarde van de current of run-unit:
- current of recordtype van het recordtype waartoe R behoort;
- current of settype van *elk* settype waarvan R owner of member is.

Ter illustratie het volgende voorbeeld.

Voorbeeld 10.7

(zie gegevens in voorbeeld 10.5)

In onderstaande tabel wordt links een programmadeel gegeven en rechts per currency-indicator wat het resultaat is van de desbetreffende opdracht. Hierbij betekent:
- : waarde currency-indicator onbekend
" : waarde currency-indicator wordt niet beïnvloed door opdracht

Programmadeel (vgl. voorbeeld 10.6)	current of						
	run-unit	recordtype			settype		
		A	B	C	SYS-A	AC	BC
MOVE 7 TO A1	-	-	-	-	-	-	-
FIND A RECORD	1.02	1.02	-	-	1.02	1.02	-
FIND FIRST RECORD OF AC SET	3.10	"	-	3.10	"	3.10	3.10
FIND OWNER OF BC SET	2.16	"	2.16	"	"	"	2.16
FIND NEXT RECORD OF AC SET	3.12	"	"	3.12	"	3.12	3.12
FIND OWNER OF BC SET	2.09	"	2.09	"	"	"	2.09

□

Met het volgende voorbeeld wordt een probleem, analoog aan dat van voorbeeld 10.4, behandeld.

Voorbeeld 10.8

Als voorbeeld 10.7, maar met een ander programmadeel.

Programmadeel	current of						
	run-unit	recordtype			settype		
		A	B	C	SYS-A	AC	BC
MOVE 7 TO A1	-	-	-	-	-	-	-
FIND A RECORD	1.02	1.02	-	-	1.02	1.02	-
FIND FIRST RECORD OF AC SET	3.10	"	-	3.10	"	3.10	3.10
FIND OWNER OF BC SET	2.16	"	2.16	"	"	"	2.16
FIND FIRST RECORD OF BC SET	3.02	"	"	3.02	"	3.02	3.02
FIND NEXT RECORD OF AC SET	3.13	"	"	3.13	"	3.13	3.13

□

Zou men in bovenstaand voorbeeld de currency-indicator van AC 'onverlet' op de waarde 3.10 willen laten staan en dus niet door de voorlaatste FIND-opdracht 'verminkt' laten worden, dan kan dit

gebeuren door in de voorlaatste FIND-opdracht gebruik te maken van het zogeheten SUPPRESS-gedeelte in de FIND-opdracht. De voorlaatste opdracht zou dan luiden:

FIND FIRST RECORD OF BC SET SUPPRESS AC

Als deze opdracht dan zou worden gevolgd door FIND NEXT RECORD OF AC SET, zou de currency-indicator van settype AC de waarde 3.12 krijgen.

In het algemeen kan met de SUPPRESS-clausule in een FIND-opdracht de verandering van currency-indicators worden 'onderdrukt', behalve de current of run-unit. Deze laatste kan nooit worden onderdrukt.

In het geval 'tweemaal achter elkaar' op hetzelfde settype 'verminking' van de currency-indicator dreigt plaats te vinden, is de zaak met de SUPPRESS-clausule 'niet te redden'. Men zal dan de (meest algemene) methode moeten volgen, waarbij gebruik wordt gemaakt van items van het type databasekey. In dit boek wordt hier niet op ingegaan. Men zij verwezen naar de specifieke DBTG-literatuur of naar een manual.

In het voorgaande hebben wij steeds verondersteld, dat elke FIND-opdracht uitvoerbaar was. Er kunnen echter redenen zijn waarom een FIND-opdracht niet uitvoerbaar is. Om de (voor ons) belangrijkste te noemen:

- met rsel: de gegeven waarde(n) van de USING-identifier(s) komt (komen) niet voor in de database;
- met rse2: bij gebruik van FIND NEXT: er is geen volgende memberoccurrence in de setoccurrence;
- gebruik van verkeerde namen: bijvoorbeeld AAC in plaats van AC.

Om niet-uitvoerbare FIND-opdrachten te kunnen signaleren (èn ook te kunnen signaleren dat een FIND-opdracht wèl uitvoerbaar is) heeft een netwerk-DBMS per programma een systeemparameter met de naam ERROR-STATUS.

Na het succesvol uitvoeren van een FIND-opdracht krijgt ERROR-STATUS de waarde 0, en anders een waarde ≠ 0. In dit laatste geval kan men aan de waarde zelf al enigszins zien 'wat er misgegaan is', bijvoorbeeld ERROR-STATUS = 0308 betekent "record- of setnaam onbekend", en ERROR-STATUS = 0307 betekent "einde van setoccurrence". Wij zullen alleen maar onderscheid maken tussen "= 0" en "≠ 0". Men bedenke verder wel, dat per programma niet meerdere ERROR-STATUS-velden beschikbaar zijn, maar precies één.

Als een FIND-opdracht succesvol is verlopen, kan men met een GET-opdracht de recordoccurrence die 'aangewezen' wordt door de current of run-unit in het werkgeheugen inlezen. De algemene vorm van een GET-opdracht is:

GET <recordname>

Voorbeeld 10.9

(zie gegevens in voorbeeld 10.5 en vergelijk met voorbeeld 10.7)

Met

```
MOVE 7 TO A1
FIND A RECORD
GET A
FIND FIRST RECORD OF AC SET
GET C
FIND OWNER OF BC SET
GET B
FIND NEXT RECORD OF AC SET
GET C
FIND OWNER OF BC SET
GET B
```

worden achtereenvolgens de volgende recordoccurrences in het werkgeheugen ingelezen (hier aangegeven met hun databasekey-waarde):

1.02; 3.10; 2.16; 3.12; 2.09 □

Wij zullen nu in de volgende paragraaf enkele van de retrievalvoorbeelden van hoofdstuk 8 behandelen met de DML van het netwerkmodel.

10.2 RETRIEVAL OP DE ZIEKENHUIS-DATABASE

In het verzamelingsmodel werden de gestelde vragen op de ziekenhuis-database op verzamelingsniveau beantwoord. Bij een netwerk-DBMS wordt dikwijls een apart (query-language) pakket aangeboden, dat behandeling van vragen op verzamelingsniveau min of meer mogelijk moet maken. De ervaring met dergelijke pakketten is nog erg gering, niet in het minst omdat de performance bij een wat grote database nogal eens te wensen overlaat.

De eigenlijke aanpak bij een netwerk-DBMS is die op recordniveau met behulp van de FIND- en GET-opdrachten.

Wij zullen nu enkele voorbeelden van hoofdstuk 8 behandelen met behulp van juist genoemde opdrachten en uitgaande van het schema van § 9.2. Wij geven de oplossingen dan in eerste instantie weer met behulp van een Pascalachtige referentietaal, zoals ook wordt gebruikt in (LUNREM). Uiteraard zijn in deze referentietaal een find- en een get-opdracht mogelijk.

Verder zullen wij daarin kunnen beschikken over de boolean functie:

end (<setname>)

Deze functie krijgt de waarde true respectievelijk false als na de uitvoering van een FIND rse2 opdracht op <setname> ERROR-STATUS = 0 respectievelijk ERROR-STATUS $\neq$ 0 is.

In tweede instantie zullen wij ook nog een enkel voorbeeld weergeven in de hostlanguage COBOL.

Voorbeeld 10.10 (vgl. voorbeeld 8.2)

Vraag

Geef naam, adres en woonplaats van de specialisten die de behandeling met code KNO6 hebben verricht.

Analyse

Wij brengen nog even het antwoord van voorbeeld 8.2 in herinnering:

$$\underline{\text{get}}\ (\{t/\{snm,sadr,swpl\} \mid t \in zkh(sp) \wedge$$
$$\exists u \in zkh(pb)[u(snr) = t(snr) \wedge u(bcd) = KNO6]\}).$$

Er blijkt dus duidelijk het volgende:
- "$t \in zkh(sp)$" geeft aan dat elk specialist-record in behandeling moet worden genomen, en daarbij geeft
- $\exists u \in zkh(pb)[u(snr) = t(snr) \wedge u(bcd) = KNO6]$ aan dat wij, vanwege "$u(snr) = t(snr)$" vanzelf in de goede richting zoeken, als wij bij elk specialist-record de bijbehorende setoccurrence van settype SP-PATBEH 'nalopen'. Dit nalopen kunnen wij overigens stoppen als wij een patiëntbehandeling-record hebben gevonden met behandelcode KNO6.

Na deze analyse moge het nu volgende antwoord duidelijk zijn. Uiteraard was een dergelijke analyse ook mogelijk geweest zonder het letterlijke antwoord van voorbeeld 8.2. In ieder geval is een verzamelingsanalyse dikwijls erg nuttig zo niet onmisbaar. Antwoorden in de referentietaal zullen wij, behoudens schema-elementen van § 9.2, in kleine letter geven en antwoorden in COBOL in grote letter.

Antwoord

```
var found: boolean
endvar;
find first record of SYS-SP set;
while not end(SYS-SP)
      do get SP; found := false;
```

```
        find first record of SP-PATBEH set;
        while not end(SP-PATBEH) and not found
              do get PATBEH;
                 if PB-BEH-CD = "KNO6"
                    then found := true
                    else find next record of SP-PATBEH set
                 fi
              od;
        if found
           then display SP-NM,SP-ADR,SP-WPL
        fi;
        find next record of SYS-SP set
     od                                                  □
```

Voorbeeld 10.11 (vgl. voorbeeld 8.6)

Vraag

Geef van elke specialist, die in 1980 meer dan tien maal de behandeling KNO7 heeft verricht: naam, adres, woonplaats, het aantal malen dat hij deze behandeling heeft verricht in 1980 en de maximale en gemiddelde duur van dit aantal.

Analyse

In de oplossing van voorbeeld 8.6 is duidelijk te zien dat per waarde van specialistnummer de verzameling patiëntbehandelingrecords met deze zelfde waarde voor specialistnummer driemaal moet worden doorlopen, en wel voor de bepaling van aantal, maximale duur en gemiddelde duur. Het aantal wordt gebruikt als onderdeel van de voorwaarde èn als onderdeel van het gevraagde. Wij kunnen dit het eenvoudigst oplossen door per specialist de *gehele* erbij behorende setoccurrence van settype SP-PATBEH door te nemen.

Antwoord

```
var maxd, somd: hoev; tel: integer
endvar;
find first record of SYS-SP set;
while not end(SYS-SP)
      do get SP; tel := maxd := somd := 0;
         find first record of SP-PATBEH set;
         while not end(SP-PATBEH)
               do get PATBEH;
                  if 800101 ≤ PB-DAT ≤ 801231 and
                     PB-BEH-CD = "KNO7"
                     then tel := tel + 1;
                          somd := somd + PB-DUUR;
                          if PB-DUUR > maxd
                             then maxd := PB-DUUR
                          fi
                  fi;
```

```
            find next record of SP-PATBEH set
        od;
    if tel > 10
        then display SP-NM,SP-ADR,SP-WPL,
                     tel,maxd,somd/tel
    fi;
    find next record of SYS-SP set
od
```

Van dit voorbeeld zullen wij hier ook de COBOL-weergave geven.

```
WORKING-STORAGE SECTION
    77    MAXD     PIC    9(4).
    77    SOMD     PIC    9(10).
    77    TEL      PIC    9(6).
    77    GEMD     PIC    9(4).

PROCEDURE DIVISION.


       FIND FIRST RECORD OF SYS-SP SET
       PERFORM VOLGENDE UNTIL ERRORSTATUS > 0.

       STOP RUN.

VOLGENDE.
       GET SP. MOVE ZERO TO TEL,MAXD,SOMD.
       FIND FIRST RECORD OF SP-PATBEH SET.
       PERFORM PB UNTIL ERROR-STATUS > 0.
       IF TEL > 10
          COMPUTE GEMD = SOMD/TEL
          DISPLAY SP-NM,SP-ADR,SP-WPL,TEL,MAXD,GEMD.
       FIND NEXT RECORD OF SYS-SP SET.

PB.
       GET PATBEH.
       IF  PP-DAT ≧ 800101 AND PB-DAT ≦ 801231 AND
           PB-BEH-CD = "KNO7"
           ADD 1 TO TEL
           ADD PP-DUUR TO SOMD
           IF PB-DUUR > MAXD
              MOVE PB-DUUR TO MAXD.
       FIND NEXT RECORD OF SP-PATBEH SET.
```
□

Tenslotte een equivalent van voorbeeld 8.11 (gebruik van sterfuncties en ext-gedeelte).

Voorbeeld 10.12 (vgl. voorbeeld 8.11)

Vraag

Geef van elke behandeling die in mei 1980 werd verricht door een specialist van de afdeling dermatologie: behandelcode en -soort, patiëntnummer en -naam, datum, specialistnummer en -naam.

Analyse

In het antwoord van voorbeeld 8.11 wordt van elk pb-tupel nagegaan of het aan de gestelde voorwaarde voldoet en zo ja, dan wordt het gevraagde met driemaal gebruik van sterfunctie met ext-gedeelte geleverd. Deze werkwijze kan hier niet worden gevolgd, omdat PATBEH geen member is van een singular set. Wij zullen hier dan ook starten vanaf de afdeling dermatologie en dan via de daarbij behorende specialisten de patiëntbehandelingen afwerken. Voldoet een patiëntbehandeling aan de voorwaarden, dan kan met o.a. enkele FIND OWNER-opdrachten het gevraagde worden geleverd.

Antwoord

```
var found: boolean
endvar;
find first record of SYS-AFD set; found := false;
while not end(SYS-AFD) and not found
      do get AFD;
         if AFD-NM = dermatologie
            then found := true
            else find next record of SYS-AFD set
         fi
      od;
if found
   then find first record of AFD-SP set;
        while not end(AFD-SP)
              do get SP;
                 find first record of SP-PATBEH set;
                 while not end(SP-PATBEH)
                       do get PATBEH
                          if 800501 ≤ PB-DAT ≤800531
                             then find owner of BEH-PATBEH set;
                                  find owner of P-PATBEH set;
                                  display PB-BEH-CD,BEH-SRT,
                                          PB-P-NR,P-NM,PB-DAT,
                                          SP-NM,SP-NR
                          fi;
                          find next record of SP-PATBEH set
                       od;
                 find next record of AFD-SP set
              od
fi
```

□

OPGAVEN

10.1 (zie gegevens in voorbeeld 10.5)
Gegeven is het volgende programmadeel:

```
MOVE 23 TO B1
FIND B RECORD
FIND FIRST RECORD OF BC SET
FIND NEXT RECORD OF BC SET
FIND FIRST RECORD OF AC SET
FIND FIRST RECORD OF SYS-A SET
FIND NEXT RECORD OF AC SET
FIND OWNER OF BC SET
```

Bepaal de waarde van de currency-indicators in de loop van het programma.

10.2 Maak een keuze uit de vragen van opgaven 8.1-8.30 en geef het antwoord in het netwerkmodel.

11 ONDERHOUD IN HET NETWERKMODEL MEMBERSHIP

11.1 ONDERHOUD IN HET NETWERKMODEL

Het netwerkmodel kent zes soorten onderhoud en dus ook zes verschillende onderhoudsopdrachten. Deze opdrachten zijn:

1. STORE	voor het *invoegen in de database* van een nieuwe recordoccurrence
2. MODIFY	voor het *veranderen* van een of meer items van een bestaande recordoccurrence in de database
3. DELETE	voor het *verwijderen uit de database* van een bestaande recordoccurrence
4. CONNECT	voor het *verbinden met een setoccurrence* van een bestaande recordoccurrence in de database
5. RECONNECT	voor het *veranderen van de verbinding met een setoccurrence* van een bestaande recordoccurrence in de database
6. DISCONNECT	voor het *ontkoppelen van de verbinding met een setoccurrence* van een bestaande recordoccurrence in de database.

Opmerking
De naam van elk van de zes onderhoudsopdrachten is in de loop der tijd (in officiële documenten en in manuals) al meermalen veranderd. In dit opzicht kunnen DBMS-pakketten van netwerksignatuur dan ook nogal verschillen vertonen. Zo is bij sommige INSERT equivalent met bovengenoemd STORE, maar bij andere is INSERT equivalent met bovengenoemd CONNECT. □

De eerste drie opdrachten hebben betrekking op de recordoccurrence zelf (reeds bekend van de bestandsorganisatie, zij het onder andere namen) en de laatste drie hebben betrekking op de verbinding van een recordoccurrence met andere recordoccurrences binnen een setoccurrence. Van deze laatste groep zullen wij overigens alleen gebruik maken van de RECONNECT-opdracht. Toelichting hierop volgt nog in § 11.2.

We zullen nu de verschillende opdrachten afzonderlijk bespreken, waarbij wij ons zullen beperken tot de belangrijkste grondvormen. Hierbij zullen wij steeds veronderstellen dat wij te doen hebben met sets waarvoor geldt SET OCCURRENCE SELECTION IS THRU CURRENT OF SET. Dit zullen wij ook veronderstellen bij onderhoudsopdrachten op de ziekenhuis-database van § 9.2, ook als in deze het schema anders luidt. Met andere vormen van de SET SELECTION-clausule kunnen onderhoudsopdrachten nogal ingewikkeld worden. Globaal gesproken wordt met de SET SELECTION-clausule weergegeven op welke wijze de geschikte setoccurrences, met name bij de STORE-opdracht, gekozen moeten worden. Door ons te beperken tot bovengenoemde vorm zal de keuze van setoccurrences steeds plaatsvinden met behulp van de currency-indicators voor de settypes.

STORE-opdracht

Deze heeft in het algemeen de vorm:

STORE <recordname>

Het effect van een succesvol uitgevoerde STORE-opdracht is:
- invoeging van de gespecificeerde recordoccurrence in de database, èn
- opname in een setoccurrence van elk settype, waarvan het desbetreffende recordtype member is, èn
- creatie van een nieuwe setoccurrence voor elk settype, waarvan het desbetreffende recordtype owner is, èn
- verandering van currency-indicators volgens de gebruikelijke regels.

Voorafgaand aan de STORE-opdracht zullen de nodige voorzieningen moeten worden getroffen ten aanzien van juiste itemwaarden van het in te voegen record en ten aanzien van juiste waarden van currency-indicators. Dit laatste is nodig om de geschikte setoccurrences beschikbaar te hebben.

Kan een STORE-opdracht niet succesvol worden uitgevoerd, dan krijgt ERROR-STATUS een waarde > 0.

Alle voorbeelden in deze paragraaf hebben betrekking op het schema van § 9.2.

Voorbeeld 11.1 (vgl. voorbeeld 8.12)

Opdracht

Voeg de volgende twee opnamerecords, als weergegeven in onderstaande tabel, in.

OPN-P-NR	OPN-R	OPN-VPR-NR	OPN-IN-DAT	OPN-UIT-DAT	OPN-SP-NR
15	DATALOGITIS	26	820614	991231	36
27	DATALOGITIS	28	820614	991231	36

Analyse

STORE kan alleen op recordniveau. Er zullen dus twee STORE-opdrachten nodig zijn.
Uit figuur 9.3 blijkt (met een oogopslag) dat OPN member is in drie settypen. Dus zal voor elk van deze drie settypen vooraf de juiste setoccurrence current gemaakt moeten worden, hetgeen met name voor het settype VPR-OPN nogal omslachtig is. Voor recordtype VPR geldt namelijk niet LOCATION MODE IS CALC. Voor elk van de beide STORE-opdrachten zal uiteraard het voorbereidende werk van dezelfde aard zijn. Wij zullen in de oplossing dan ook werken met een procedure voeginopn.

Oplossing (in referentietaal)

```
               procedure voeginopn (var pnr    : integer;
                                        opnr   : charstring;
                                        vnr    : integer;
                                        indat  : integer;
                                        uitdat : integer;
                                        snr    : integer)
                           begin var setp,setsp,setvpr: boolean
                                 endvar;
                                 P-NR := pnr; find P record;
% juiste setoccurrence           if ERROR-STATUS > 0
  zoeken van P-OPN %                then setp := false
                                    else setp := true
                                 fi;
                                 SP-NR := snr; find SP record;
% juiste setoccurrence           if ERROR-STATUS > 0
  zoeken van SP-OPN %               then setsp := false
                                    else setsp := true
                                 fi;
                                 find first record of SYS-AFD set;
                                 setvpr  := false;
                                 while not end(SYS-AFD) and not setvpr
                                       do find first record of AFD-VPR set;
                                          while not end(AFD-VPR) and not setvpr
                                                do get VPR;
                                                   if VPR-NR = vnr
% juiste setoccurrence                                then setvpr := true
  zoeken van VPR-OPN %                                else find next record of AFD-VPR set
                                                   fi
                                                od;
                                          if not setvpr
                                             then find next record of SYS-AFD set
                                          fi
                                       od;
```

```
                          if setp and setsp and setvpr
                             then OPN-P-NR := pnr;
                                  OPN-R := opnr;
% waarde van in te                OPN-VPR-NR := vnr;
  voegen record %                 OPN-IN-DAT := indat;
                                  OPN-UIT-DAT := uitdat;
                                  OPN-SP-NR := snr;
% invoegen %                      store OPN
                             else display setp,setsp,setvpr
                          fi
% einde procedure%     end;
                       voeginopn (15, "DATALOGITIS", 26, 820614, 991231, 36);
                       voeginopn (27, "DATALOGITIS", 28, 820614, 991231, 36)
```

□

MODIFY-opdracht

Deze heeft in het algemeen de vorm

MODIFY <recordname> <datanamelist>

Het effect van een succesvol uitgevoerde MODIFY-opdracht is de verandering van de velden in <datanamelist> van het gespecificeerde record in vooraf opgegeven waarden.

Ook hier geldt weer dat bij niet succesvolle uitvoering ERROR-STATUS een waarde > 0 krijgt

Voorbeeld 11.2 (vgl. voorbeeld 8.14)

Opdracht

Verander in het opnamerecord met patiëntnummer 15 en indat 820614 de uitdat in 820625.

Analyse

Aangezien OPN geen LOCATION MODE IS CALC heeft, moet het bedoelde opnamerecord gezocht worden via een setoccurrence van P-OPN.

Oplossing

```
var found: boolean endvar;
P-NR := 15;
find P record;
if ERROR-STATUS > 0
   then display "patiënt niet bekend"
   else find first record of P-OPN set; found := false;
        while not end (P-OPN) and not found
              do get OPN;
```

```
                    if OPN-IN-DAT = 820614
                       then found := true
                       else find next record of P-OPN set
                    fi
                 od;
           if found
              then OPN-UIT-DAT := 820625
                   modify OPN OPN-UIT-DAT
              else display "opname onbekend"
           fi
   fi
```
□

Bij verandering op meerdere records zijn ook meerdere MODIFY-opdrachten nodig.

Voorbeeld 11.3 (vgl. voorbeeld 8.16)

Opdracht

Verminder van elke specialist met een aantal bedden > 0, dit aantal met 1.

Oplossing

```
   find first record of SYS-SP set;
   while not end(SYS-SP)
         do get SP;
            if SP-AB > 0
               then SP-AB := SP-AB - 1
                    modify SP SP-AB
            fi;
            find next record of SYS-SP set
         od
```
□

DELETE-opdracht

Deze heeft in het algemeen de vorm

DELETE <recordname>

Het effect van een succesvol uitgevoerde DELETE-opdracht is:
- ontkoppeling van alle setoccurrences, waartoe het gespecificeerde record als member behoort;
- verwijdering van genoemd record uit de database; dit record is het current record of run-unit;
- current of run-unit krijgt onbepaalde waarde (overige currency-indicators veranderen niet).

Een DELETE-opdracht is niet uitvoerbaar als het gespecificeerde record owner is van een niet-lege setoccurrence.

Voorbeeld 11.4 (vgl. voorbeeld 8.13)

Opdracht

Verwijder de patiëntbehandelingstupels, die betrekking hebben op de patiënt met patiëntnummer 82 en op een behandelingsdatum in 1980.

Analyse

PATBEH bezit geen LOCATION MODE IS CALC. Dus ook hier weer via een setoccurrence en wel van settype P-PATBEH.

Oplossing

```
P-NR := 82;
find P record;
if ERROR-STATUS = 0
  then find first record of P-PATBEH set;
       while not end(P-PATBEH)
             do get PATBEH;
                if 800101 ≦ PB-DAT ≦ 801231
                   then delete PATBEH
                fi;
                find next record of P-PATBEH set
             od
fi
```
□

In voorbeeld 8.15 staat de volgende opdracht: Verander adres, woonplaats van specialist 12 in dalweg 18, geldrop en verander zijn afdelingsnummer in 7.

Na deze verandering zou het desbetreffende specialistenrecord met een verkeerde setoccurrence van settype AFD-SP zijn verbonden, namelijk met die setoccurrence, waarvan zijn oude afdeling owner is. Om dit te corrigeren is een RECONNECT-opdracht nodig.

Hiermee zijn wij aangeland bij de tweede soort onderhoudsopdrachten, namelijk die waarbij de inhoud van een recordoccurrence niet verandert, maar wel het verband met andere recordoccurrences via verandering van setoccurrence.

RECONNECT-opdracht

Deze heeft in het algemeen de vorm:

RECONNECT <recordname> WITHIN <setname> SET

Het effect van een succesvol uitgevoerde RECONNECT-opdracht is dat het gespecificeerde record (zijnde het current record of run-unit) voor het gespecificeerde settype ontkoppeld wordt van de setoccurrence, waarvan het nu een member is, en wordt toegevoegd aan de gespecificeerde setoccurrence van hetzelfde settype.

Voorbeeld 11.5 (vgl. voorbeeld 8.15)

Opdracht

Verander adres, woonplaats van specialist 12 in dalweg 18, geldrop en verander zijn afdelingsnummer in 7.

Analyse

Na verandering van het desbetreffende specialistrecord moet een RECONNECT-opdracht plaatsvinden om verbinding met de juiste setoccurrence tot stand te brengen.

Oplossing

```
var found: boolean endvar;
SP-NR := 12;
find SP record;
if ERROR-STATUS > 0
   then display "specialist 12 onbekend"
   else SP-ADR := "DALWEG 18"
        SP-WPL := "GELDROP"
        SP-AFD-NR := 7
        modify SP SP-ADR,SP-WPL,SP-AFD-NR;
        find first record of SYS-AFD set;
        found := false;
        while not end(SYS-AFD) and not found
              do get AFD;
                 if AFD-NR = 7
                    then found := true
                    else find next record of SYS-AFD set
                 fi
              od;
        if not found
           then display "afdeling 7 onbekend"
           else reconnect SP within AFD-SP set
        fi
fi
```
□

Dat de andere twee onderhoudsopdrachten voor opname in of verwijdering uit setoccurrences (CONNECT of DISCONNECT) niet zijn toegestaan op het schema van § 9.2 hangt samen met de keuze van zogeheten memberships in dit schema. Het begrip membership wordt in de volgende paragraaf behandeld.

11.2 MEMBERSHIP

In de beschrijving van elk van de 19 settypen in het schema van § 9.2 komt de volgende regel voor:

AUTOMATIC MANDATORY   of   MANDATORY AUTOMATIC.

De volgorde van de woorden AUTOMATIC en MANDATORY doet overigens niet ter zake.

AUTOMATIC bepaalt hoe de verbinding van een nieuwe recordoccurrence met een setoccurrence tot stand dient te komen, namelijk: een STORE-opdracht heeft automatisch een CONNECT-opdracht tot gevolg zonder dat deze laatste opdracht gegeven wordt.

Voor het eerste-verbinding-voorschrift is behalve AUTOMATIC ook MANUAL mogelijk. In dit laatste geval kan de eerste verbinding met een setoccurrence uitsluitend tot stand komen met een CONNECT-opdracht.

MANDATORY bepaalt wat met eenmaal verbonden recordoccurrences mag en moet gebeuren, namelijk: een eenmaal verbonden recordoccurrence moet verbonden blijven, maar niet per se met dezelfde setoccurrence.

Voor het behoud-verbinding-voorschrift is behalve MANDATORY ook OPTIONAL mogelijk. Dit laatste houdt in dat een eenmaal verbonden recordoccurrence niet verbonden hoeft te blijven.

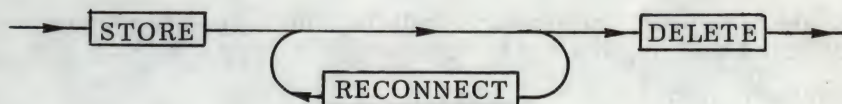
De volgende vier combinaties zijn dus mogelijk:

1. AUTOMATIC   MANDATORY
2. AUTOMATIC   OPTIONAL
3. MANUAL   MANDATORY
4. MANUAL   OPTIONAL

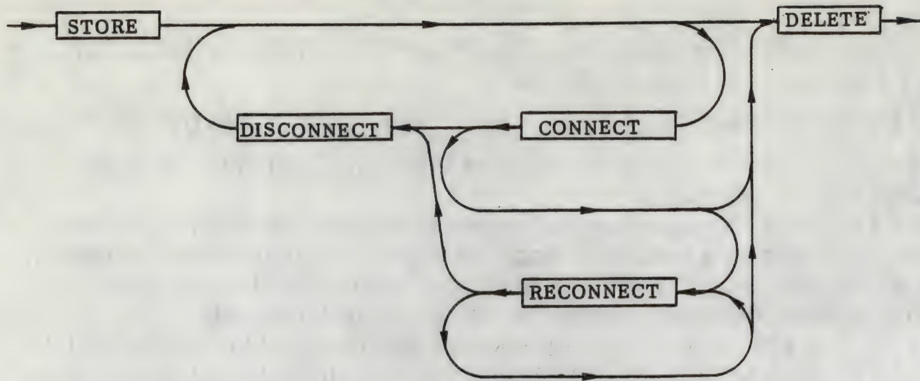
Met het bovenstaande ligt vast welke verbindingsopdrachten (CONNECT, DISCONNECT, RECONNECT) en in welke volgorde zijn toegestaan op een member-recordoccurrence afhankelijk van de gekozen membership-combinatie.

Wij zullen dit toelichten met een soort 'syntax-diagram' voor de combinaties 1 en 4.

Ad 1



Ad 4



In feite kan worden volstaan met de eerste membership-combinatie, mits uitsluitend gewerkt wordt met genormaliseerde databases. Dat is ook de reden waarom in het schema van § 9.2 alleen deze eerste combinatie voorkomt.

OPGAVEN

11.1 Voer de volgende opdrachten uit op de database van voorbeeld 10.5.
- a. Voeg in het A-record met A1 = 17 en A2 = 7.
- b. Voeg in het C-record met C1 = 1, C2 = 5 en C3 = 49.
- c. Verander B2 van het B-record met B1 = 9 in de huidige waarde +5.
- d. Verminder C3 met 5 van elk C-record, waarvoor geldt C3 > 20.
- e. Verwijder het C-record met C1 = 2 en C2 = 5.

De opgaven 11.2 tot en met 11.6 hebben betrekking op het schema van § 9.2 (vgl. opgaven 8.34-8.38).

11.2 Voeg de patiëntbehandelingsrecords in, die zijn weergegeven in de tabel van opgave 8.34.

11.3 Als opgave 8.35.

11.4 Als opgave 8.36.

11.5 Verwijder het patiëntbehandelingsrecord betreffende behandeling DL11 op patiënt 15 op 16 juni 1982.

11.6 Verwijder het specialistrecord van specialist 8 en alle patiënt-behandelings- en medicijnverstrekkingrecords, waarin deze specialist voorkomt.

11.7 Geef de 'syntax-diagrammen' voor de membership-combinaties 2 en 3.

LITERATUUR

AANGEHAALDE LITERATUUR

(BEM) BEMELMANS, T.M.A., *Bestuurlijke informatiesystemen en automatisering*, Stenfert Kroese, Leiden 1981, 236 blz.
In dit boek wordt onder meer uitgebreid ingegaan op het nauwe verband tussen informatie en beslissing.

(BROCK) BROCK, E.D. de, *Verzamelingen en databases*, in APERS, P.M.G. (red.): *Colloquium Databankorganisatie, deel 1*, blz. 89-116, Mathematisch Centrum, Amsterdam 1981.
Goed achtergrondartikel voor deel II.

(CODASYL 69)
A survey of generalized database management systems, CODASYL Systems Committee, May 1969.

(CODASYL 71)
April 71 Report, CODASYL Data Base Task Group, 1971, 267 blz.

(CODD 70) CODD, E.F., *A relational model of data for large shared data banks*, Communications of the ACM Vol.13, no.6, June 1970, pp.377-387.
Dit is het welbekende (en welgelezen?) artikel van Codd, waarmee de 'zegetocht' van het relationele model is begonnen.

(CODD 79) CODD, E.F., *Extending the database relational model to capture more meaning*, ACM Transactions on database systems Vol.4, no.4, December 1979, pp.397-434.
Een jammer genoeg vrijwel onleesbaar artikel, dat blijkbaar bedoeld is om een synthese tot stand te brengen tussen de onderzoekresultaten van de zeventiger jaren. Er wordt echter wel veel naar gerefereerd!

(DATE) DATE, C.J., *An introduction to database systems*, 3rd edition, Addison Wesley 1981, 574 blz.
Het meest bekende leerboek op het gebied van databases. Bevat ook een grote hoeveelheid, van notities voorziene, literatuurverwijzingen.

(DOROVK) DOOREN, R. van, D. OVERKLEEFT & E.J. RUFF, *Gegevensbanken*, Kluwer 1979, 224 blz.
3e ongewijzigde druk.

(DORSTR) DOOREN, R. van & Th.G. STRENG, *Programmeertalen voor gegevensbanken*, 2e druk, Kluwer 1981, 205 blz.

De boeken (DOROVK) en (DORSTR) geven geen algemene behandeling van databases, maar richten zich uitsluitend op het netwerkmodel, waarbij het eerste de structurering behandelt en het tweede het gebruik en onderhoud.

(LUN-REM) LUNBECK, R.J. & F. REMMEN, *Bestandsorganisatie*, 5e druk, Academic Service, Den Haag 1982, 210 blz.
Het boek bevat een systematische behandeling van de verschillende vormen van bestandsorganisatie en is als zodanig een goede basis om een duidelijk begrip te krijgen van de implementatieproblemen van databases.

(MARTIN) MARTIN, J., *Computer database organization*, 2nd ed., Prentice Hall 1977.
Goed leesbaar, prettig uitgevoerd boek over databases. Is oppervlakkig en bevat onnodige herhalingen.

(OLLE) OLLE, T.W., *The Codasyl approach to data base management*, Wiley, New York 1978.
Dit boek geeft een goed overzicht van het netwerkmodel en gaat daarbij de (historisch ontstane) onduidelijkheden niet uit de weg. Ook enkele niet-netwerksystemen worden behandeld.

(SMITH) SMITH, J.M. & D.C.P. SMITH, *Principles of database conceptual design*, Proc. NYU Symposium on database design, New York 1978, pp.35-49.
Het echtpaar Smith doet veel onderzoek op het gebied van logische gegevensstructuren. Zij komen daarbij met afwisselend waardevolle en onbruikbare ideeën.

NIET-AANGEHAALDE LITERATUUR

DAVIS, B. *The selection of database software*, NCC, Manchester 1977, 328 blz.
Interessante en uitgebreide bepsreking van zes Data Base Management Systemen: ADABAS, IDMS, IMS, ROBOT, S2000, TOTAL. Wel wat duur (ƒ 236,--).

DEEN, S.M. *Fundamentals of data base systems*, MacMillan Press 1977, 224 blz.
Heel aardig, inleidend boekje over databases.

MOHAN, C. *An overview of recent database research*, Database, quarterly newsletter of SIGBDP of the ACM, Vol.10, no.2, 1978, pp.3-24.
Zeer uitgebreid literatuuroverzicht van 415(!) boeken/tijdschriftartikelen.

REGISTER

ACADEMIC SERVICE INFORMATICA UITGAVEN

INLEIDINGEN

'Basiskennis Informatieverwerking' van Jan Everink

'Computers en de cybernetische samenleving' van M.A. Arbib

'De viewdata revolutie' van S. Fedida en R. Malik

MICROCOMPUTERS

'TRS-80 Basic: Een gids voor zelfstudie' van B. Albrecht e.a.

'CP/M: een gids voor zelfstudie' van J.N. Fernandez en R. Ashley

PROGRAMMEREN EN PROGRAMMEERTALEN

'Inleiding tot het programmeren, deel 1' van ir. J.J. van Amstel e.a.

'Inleiding tot het programmeren, deel 2' van ir. J.J. van Amstel e.a.

'Programmeren, deel 1: Inleiding' van prof.drs. C. Bron

'Programmeren, deel 2: Van analyse tot algoritme' van prof.drs. C. Bron

'Inleiding programmeren' van prof.dr. R.J. Lunbeck

'Inleiding datastructuren' van prof.dr. R.J. Lunbeck

'Inleiding programmeren en programmeertechnieken', EIT-serie, deel 1

'Het Groot Pascal Spreukenboek' van H.F. Ledgard, P.A. Nagin en J.F. Hueras

'Basic', EIT-serie, deel 3

'Cursus eenvoudig Pascal' van prof.dr. A. van der Sluis en drs. C.A.C. Görts

'Cursus Pascal' van prof.dr. A. van der Sluis en drs. C.A.C. Görts

'Inleiding programmeren in Pascal' van C. van de Wijgaart

'Cursus Fortran 77' van J.N.P. Hume en R.C. Holt

'Cursus COBOL' van A. Parkin

'Cursus Algol 60' van prof.dr. A. van der Sluis en drs. C.A.C. Görts

SYSTEEMPROGRAMMATUUR

'Computersystemen' van prof.ir. D.H. Wolbers

'Bedrijfssystemen', EIT-serie, deel 4

'Systeemprogrammatuur' van drs. H. Alblas

'Vertalerbouw' van drs. H. Alblas e.a.

DATASTRUCTUREN, BESTANDSORGANISATIE, DATABASE

'Bestandsorganisatie' van prof.dr. R.J. Lunbeck en drs. F. Remmen

'Gegevensstructuren' van R. Engmann e.a.

'Informatiestructuren, bestandsorganisatie en bestandsontwerp', EIT-serie, deel 5

INFORMATIEANALYSE, SYSTEEMONTWERP

'Eerlijk en helder', van prof.dr. P.G. Bosch

'Voorbereiding van computertoepassingen' van prof. A.B. Frielink

'Analyse van informatiebehoeften en de inhoudsbeschrijving van een databank' van prof.dr. P.G. Bosch en ir. H.M. Heemskerk

'IBSEN - Een SIMULA programma voor gebruik bij de beschrijving van informatiebehoeften' van prof.dr. P.G. Bosch en ir. H.M. Heemskerk

'Systeemontwikkeling volgens S.D.M.' van H.B. Eilers

'Een samenvatting van de System Development Methodology SDM' van PANDATA

'Gegevensanalyse' van R.P. Langerhorst

'Cases op het gebied van administratieve organisatie en informatieverzorging (inclusief systeem-ontwerp)' van prof.dr. P.G. Bosch en H.A. Te Rijdt

'Uitwerkingenboek bij cases op het gebied van administratieve organisatie en informatieverwerking' van prof.dr. P.G. Bosch en H.A. Te Rijdt

'SIMULATIE, een moderne methode van onderzoek' van drs. S.K.T. Boersma en ir. T. Hoenderkamp

THEORETISCHE INFORMATICA

'Abstracte automaten en grammatica's' van prof.dr. A. Ollongren en ir. Th.P. van der Weide

AANVERWANTE ONDERWERPEN

'Lineaire programmering als hulpmiddel bij de besluitvorming' van drs. S.W. Douma

INFORMATIE OVER DEZE PUBLIKATIES BIJ:

Academic Service
Postbus 96996
2509 JJ 's-Gravenhage
(Tel. 070-247238)